Ph.D Theis by Yao Liu 2005

The Privacy Preservation of Data Cubes

By

Yao Liu,

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Doctor of Philosophy

National University of Singapore

Doctor of Philosophy (2005) National University of Singapore
(Department of Computer Science) Kentridge, Singapore

TITLE: The Privacy Preservation of Data Cubes

AUTHOR: Yao Liu,

SUPERVISOR: Sung Sam Yuang

NUMBER OF PAGES: xiii, 87

Abstract

A data warehouse stores current and historical records consolidated from multiple transactional systems. On-Line Analytical Processing (OLAP) is concerned with the analysis of huge volumes of data collected in a data warehouse and provided multidimensional views for analyzers and managers to gain insight into the performance of the enterprise. With the development of OLAP, users are not only inner analysts or managers but also customers, partners or even third parties. This makes the privacy preservation for data cubes become more and more important.

In this thesis, we proposed two simple but effective methods to protect sensitive individual data of data cubes. These approaches were motivated by the following observation: a single data item in a data cube is not likely to be accessed alone, but a number of data are often aggregated to give summarized information and the trends of database. This is the reason why range queries are heavily used in data cubes. Therefore we restricted our research on the two most important range queries: range-sum query and range-max query. The goal was to closely estimate the answer for range queries without revealing individual data.

Since response time of OLAP application is critical even for very large range queries, we adopted the basic idea of random data distortion method in Statistical Databases. Because the random data distortion method was simple and no overhead would be introduced to the query processing. However, unlike the random data distortion method, our methods added random noise to original data in a constraint way so that the effect of the noise could be counterbalanced as much as possible. As a result, the individual data were distorted and looked very different from the original data, but the summation of a range of distorted data was almost remained the same as that of the original data.

The experiments were done on APB benchmark data set from OLAP council. The results showed that our methods achieved both better privacy preservation and better range query accuracy for range queries than traditional random data distortion alternatives.

Acknowledgements

I thank my supervisor Dr. Sung Sam Yuan for the three years of wonderful advice and support. He gave me both the technical knowledge and more importantly the example of academic integrity in conducting research. His comments, advices and ideas were invaluable throughout the entire project.

I also thank my parents who shoulder most of the housework for me so that I can concentrated on my research.

Publications

1. Yao Liu, Sam Y. Sung, Hui Xiong, Peter Ng, Data Declustering with Replication, in Proc. of the 9th International Conference on Database Systems for Advanced Applications (DASFAA 2004), pp. 682 - 693, Korea, 2004.

2. Sam Y. Sung, Yao Liu, Pter Ng, Privacy Preservation for Data Cubes, Proceedings of the 20th Internatinal Conference on Data Engineering, pp.826, 2004 (short paper)

3. Sam Y. Sung, Yao Liu, Hui Xiong, Peter Ng, Privacy Preservation for Data Cubes, Knowledge and Information Systems - An International Journal (KAIS), accepted as a regular paper, 2005.

4. Yao Liu, Sam Y. Sung, Hui Xiong, A Cubic-Wise Balance Approach for Privacy Preservation in Data Cubes, Information Sciences, accepted as a regular paper, 2005.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

A *data warehouse* defined in [1] is a collection of data from multiple sources, integrated into a common repository and extended by summary information (such as aggregate views) that is primarily used in organizational decision making. A class of queries that typically involves group-by and aggregation operators is called On-Line Analytical Processing or OLAP. OLAP software enables analysts and managers to gain insight into the performance of an enterprise and help decision making.

1.1.1 OLAP introduction

OLAP applications present the end user with information rather than just data. They make it easy for users to identify patterns or trends in the data very quickly, without the need for them to search through mountains of "raw" data. Typically this analysis is driven by the need to answer business questions such as "How are our sales doing this month in North America?" From this foundations, OLAP applications move into areas such as forecasting and data mining, allow users to answer questions such as "what are our predicted costs for next years?" and "Show me our most successful salesman".

OLAP is implemented in a multi-user client/server mode and offers consistently rapid response to queries, regardless of database size and complexity. OLAP helps the user synthesize enterprise information through comparative, personalized viewing, as well as through analysis of historical and projected data in various "what-if" data model scenarios. This is achieved through use of an OLAP Server.

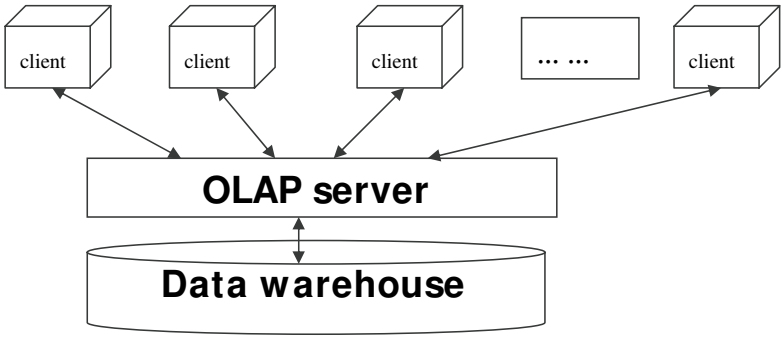


Figure 1.1: OLAP Client/server mode.

An OLAP server is a high-capacity, multi-user data manipulation engine specifically designed to support and operate on multi-dimensional data structures. A multi-dimensional structure is arranged so that every data item is located and accessed based on the intersection of the dimension members which define that item. The design of the server and the structure of the data are optimized for rapid ad-hoc information retrieval in any orientation, as well as for fast, flexible calculation and transformation of raw data based on formulaic relationships.

The OLAP Server may either physically stage the processed multi-dimensional information to deliver consistent and rapid response times to end users, or it may populate its data structures in real-time from relational or other databases, or offer a choice of both. Given the current state of technology and the end user requirement for consistent and rapid response times, staging the multi-dimensional data in the OLAP Server is often the preferred method.

1.1.2 OLAP models

OLAP applications are dominated by ad hoc, complex queries. There are two approaches(data models), Relational-OLAP(ROLAP) and Multidimensional-OLAP (MOLAP)[3], also known as *data cube*[2].

Figure1.2 is the multidimensional view of table 1.1.

Table 1.1: 2-dimensional view of data

pic	timeid	locid	sales
11	1	1	25
11	2	1	8
12	1	1	30
12	2	1	20
13	1	1	20
13	2	1	40
11	1	2	35
11	2	2	22
12	1	2	10
12	2	2	26
13	1	2	45
13	2	2	20

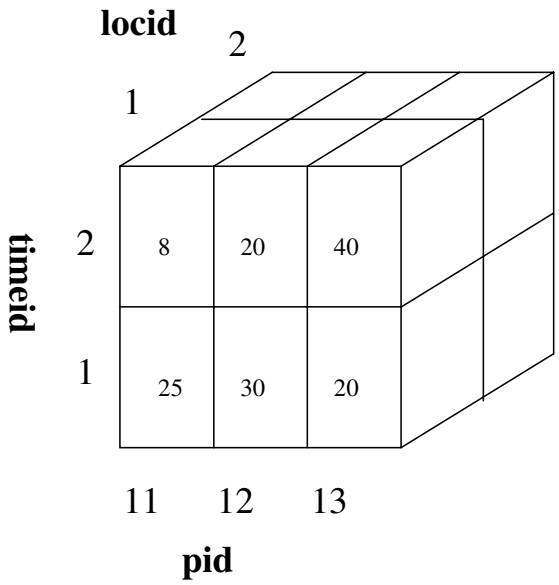


Figure 1.2: multidimensional view of data.

Data cube systems support a query style in which the data is best thought of as a multidimensional array, which is influenced by end user tools such as spreadsheets, in addition to database query language. In the data cube model, a data cube is constructed from a subset of attributes in the database. Certain *attributes* are chosen to be *measure attributes*, i.e., the attributes whose values are of interest. The remaining attributes are referred to as *functional attributes* or *dimensions*. The measure attributes of records with the same functional attributes values are combined (i.e. summed up) into aggregate values. Thus, a data cube could be viewed as a d-dimensional array, indexed by the values of the d dimension attributes, whose cells contain the values of the measure attributes for the corresponding combination of dimension attributes.

1.1.3 OLAP operations

OLAP functionality is characterized by dynamic multi-dimensional analysis of consolidated enterprise data supporting end user analytical and navigational activities including:

1. calculations across dimensions, through hierarchies and/or across members
2. trend analysis over sequential time periods
3. slicing subsets for on-screen viewing
4. drill-down to deeper levels of consolidation rotation to new dimensional comparisons in the viewing area

The detail of OLAP operations is presented in Chapter1.

1.1.4 Range query

Range query is one of the most important queries of data cube. Range queries apply a given aggregation operation over selected cells where the selection is specified as contiguous ranges in the domains of some of the attributes. Two types of queries, *Range-sum* and *Range-max*, have been extensively studied in recent years [4, 5]. Range-sum (Range-max) query is finding the summation values (the maximum values) over selected cells of a data cube where the selection is specified by a range of contiguous values for each dimension. For example, finding the total sales of stationary items, which has an item code ranging from 1201 to 1300, between day 130 and 159 in the western outlets, with branch-no ranging from 45 to 89 is a range-sum query. It can be realized using the following SQL-type statement:

select sum(amount) **from** sales
where $(1201 \leq item \leq 1300)$ and $(130 \leq day \leq 159)$ and $(45 \leq branch \leq 89)$

Marginal sum is a special case of range sum query.

For example, consider a data cube from insurance company. Assume the data cube has four dimensions: age, year, state, and type. Further assume that the domain of each dimension is as follows:

age(1..100); year(1987..1998); state(0..49); type(auto, home, health). The size of data cube is $100 \times 12 \times 50 \times 3$.

Assume the measure attribute is revenue. Q1: $< age = all, year = 1995, state = all, type = auto >$(Q1$< *, 1995, *, auto >$)

To answer query of type Q1 efficiently, the domain of each dimension be augmented with an additional value 'all', denoted by '*', to store aggregated value of the measure

attribute in all of the cells along that dimension. In Q1 above, the data cube will be extended to $101 \times 13 \times 51 \times 4$, e.g. this query is actually corresponding to a cell of the extended cube. Such cell with '*' is called marginal point and the value of such cell is called marginal sum in this thesis.

1.1.5 OLAP privacy

Data warehouse is primarily built as an open system to support OLAP, which requires the open nature of data warehouse. However, with the growth of OLAP, the range of data warehouse users is steadily growing up to include customers, partners or even third parties. This leads to privacy concern [6] and the needs of proper access control policies. Indeed, inappropriate disclosure of sensitive data stored in the underlying data warehouses results in the breach of individual's privacy and jeopardizes the organization's interest. It is well recognized that access control alone is insufficient in controlling information disclosure, because information not released directly may be inferred indirectly by manipulating legitimate queries about aggregated information.

In OLAP application, analysts are rarely interested in individual data, which is often sensitive, but care more about summary data from aggregate operations [7]. So preserving the confidential information in individual data cells while still being able to provide an accurate estimation of aggregations is the main focus of this paper.

1.2 Literature review

Compared with researches on accelerating response time in OLAP [4][5] [8][9][10][11][12], OLAP privacy receives little attention. Related literature can be grouped into five categories. The first is an introduction of a summary of security-control in statistical databases. The second addresses privacy preserving data mining. The third is about

data access control. In addition, the fourth category is about the idea of Quasi-cube. Finally, a fifth category briefly describes cardinality-based security control in data cubes.

1.2.1 Security control in statistical databases

Data privacy is a concern common to both OLAP and Statistical Database (SDB) communities. The similarities and differences between OLAP and SDB were given in detail in [13]. The similarities are that both OLAP and SDB deal with multidimensional datasets, and both are concerned with statistical summarizations over the dimensions of the data set.

In contrast to the area of OLAP, there has been extensive research in the statistical databases community on the privacy problem (an excellent survey is in [14]).

There were mainly two techniques proposed by the statistical databases community:

1. *query restriction* – The query restriction category included restricting the size of query results [15, 16], and controlling the overlap of successive queries [17].

2. *data perturbation* – The data perturbation category included swapping values between records [18], swapping attribute values [19, 20], replacing the original database with a sample from the same distribution [5], adding noise to the values in the database [21], adding noise to the results of a query [22], and sampling the results of a query [23].

Since our goal is to allow data access with minimum restriction, query restriction methods are undesirable. Furthermore, data security may still be in danger from a set of smartly designed probes accessing the data cube.

Because of the fast response requirement and distributed environment of our model, we know that accessing the original data cube directly is not feasible. Therefore, many of the data perturbation methods that are based on the use of the original data cube or queried results are also not feasible.

Notice that the functional attributes do not usually need protection, and as their attribute values order needs to be preserved, swapping techniques are undesirable and unnecessary.

The fixed-data perturbation for numerical attributes (or called *value distortion*) in SDB may be useful for OLAP privacy preservation. This method was developed by [21]. The idea was to return a perturbed value by adding a random noise drawn from some distribution to the true value. Suppose, for example, that the true value of a given attribute (e.g. sales) of an entity k is x_k. The answer to the sum query, under this method, will be answer $= \sum_{k=1}^{n} y_k$ where $y_k = x_k + z_k$, z_k is a random-perturbation variable with $E(z_k) = 0$ and $Var(z_k) = \sigma^2$, and z_k are independent for different k's.

1.2.2 Privacy-preserving data mining

"Privacy preserving data mining" [24] means getting valid data mining results without learning the underlying data values. In most applications, the privacy issue is somehow related to an individual or groups of individuals sharing some common characteristics in a given context. Sometimes the "patterns" detected by a data mining system may be used in a counter-productive manner that violates the privacy of an individual or group of individuals. Therefore, it is important to protect the privacy of the data and its context while mining.

Privacy preserving data mining has drawn considerable attentions [25][26][27][28][29][30]. The main idea was to scramble a customer's personal data by randomization and simultaneously reconstructing the original distributions of the values of the confidential attributes. Note that the goal was to reconstruct distributions, not values in individual records. This way, both the objectives of privacy protection and statistical based rule accuracy could be achieved at the same time. In [24], it has been shown that it was possible in decision-tree classification. Their work was later also extended to association rule [26]. The work in [31] addresses the problem of association rule mining where transactions were distributed across sources.

Privacy preservation in data mining concentrates on the reconstruction of *data distribution*, whereas our concern is the reconstruction of *aggregates*. The underlying database used by data mining is not necessarily a data cube. Sparsity and response time are more critical in our problem, and distribution sensitivity is not a concern. Therefore, the methods developed for data mining are not adaptable because they are different application domains with different concerns and working environments.

1.2.3 Data access control

Access control is an important security technique and is commonly used in operational data sources to control the access to data sources (data warehouse and source databases). However, the relational model predominates operational systems while OLAP systems make use of the non-traditional multidimensional model. In OLAP systems, protection is not defined in terms of tables, but in terms of dimensions, hierarchical paths, and granularity levels [32]. Thus it is not easy to map traditional access control in OLAP systems.

In [32], different OLAP access control requirements were proposed. These requirements were related to OLAP operations mentioned in the previous section. The main objective of access control was to hide information in cubes. The advanced requirements satisfied more security requirements, but also created more difficulties in implementation because of the increased complexity.

With complex security requirements, information hiding would cause several problems. For example, if certain slices are hidden, how should the higher level summary data be decided? Should the hidden slices be included? If including the hidden slices to reflect the 'real' totals, the report will be left in an inconsistent state (the displayed total is not the summary of displayed data). If the hidden slices are not included, only the total of the displayed values being shown, then the total will have to change from one query to another even though the queried domain has remained the same. In fact, both approaches can be found in today's commercial systems.

In OLAP applications, complex access control is too slow. In addition, allowing outsiders to directly access the inner database may result in the leakage of sensitive information even with perfect access control. For example, some companies allow their customers or partners to access their data cube for viewing aggregated data, such as monthly or yearly data, but the individual data is very sensitive and should be strictly protected.

1.2.4 Quasi-cube

In the OLAP literature, there was work on approximating queries on sub-cubes from higher-level aggregations in [7]. The idea was to divide the cube into regions and to use a statistical model to describe each region with an additional estimation procedure being introduced to estimate the missing entries with a certain level of

accuracy based on the incomplete specified cube (the quasi-cube). The quasi-cube was designed to save storage, but it did hide some individual entries. Queries of the quasi-cube could provide approximate answers by estimating the missing entries with a reasonable level of accuracy using linear regression.

However, the estimation procedure must be faster than computing the data from the underlying relations and a certain portion of the storage has to be used for the description.

1.2.5 Cardinality-based security control

Recently, Wang et al [33] proposed a cardinality-based security control in data cubes. By exploring special structures of data cube operator, such as group-by, cross-tab and sub-total, the author derived cardinality-based sufficient conditions for securing data in data cubes. The main idea of this method was to enforce query restriction. Only those queries, which were regarded as safe, would be answered.

Such a query restriction method can protect the data cube from information leakage through repeating queries. However, due to the query restriction, some legal queries may not satisfy the specified security requirements. As a result, these queries have to be denied or modified for resubmission. Furthermore, the overhead of judging whether a query is safe or not is unavoidable. As we know, the response time is critical in OLAP systems and any extra procedures can potentially slow down the response time.

1.3 Project objectives

Data Privacy for data warehouses/data cubes is important for companies. An investment company can collect and hold the following data and information:

- Over a million customer accounts.
- In each account, the stocks a customer bought, and the quantity, the purchase date and the price of each purchase.
- The stocks each customer sold, and the quantity, the sale date and the price.

Much interesting and useful knowledge is contained in such an investment data warehouse but individual investors will lose trust in the company if their data is revealed. The company may be willing to participate in a collaboration project, but only if the protection of its own data can be ensured.

The following four features make our problem different from other applications in data privacy preservation.

- Our model is a client/server model in which the server is the information holder and the clients are data cube users. In other applications, the clients are closer to the server while our application is more like a distributed model.
- Since data cube is an online tool, the response time is critical, so data needs to be accessed as quickly as possible. Therefore, data cubes are materialized and possibly pre-computed on the server site and delivered to the client sites.
- The only attributes that need to be protected are the measure attributes.
- Analysts seldom interest in individual data but aggregated information.

Typically, people think of the issue of privacy as the need to protect individual data. Most effort has been in this direction. In a data cube model, this privacy problem can be scaled up to aggregated or *collective* information. That is, the release of a block (sub-cube) of information should not enable *identification* of cell information. Such "collective information" is generally needed in the decision making process, but its release should not cause any concern. Therefore, the objective of privacy preserving for data cubes is to develop a technique that guarantees no cell data being revealed when collective information is released.

More specifically, the technique should fulfill the three goals:

1. Security - Any sensitive data must be protected from being released

2. Accuracy - The results of any analysis are accurate enough for making business decision.

3. Accessibility - Since the response time is critical, the data access should be as easy as possible. That is there should be no unnecessary restriction or control on the access of data.

1.4 Contributions

The main contributions of this thesis are summarized below.

1. We propose two simple but effective solutions, called the zero-sum method and cubic-wise balance method, for providing an accurate estimation of the original values for range queries in a data cube while preserving the confidential information in individual data cell. Our methods are based on random data distortion techniques and relative random data distortion is applied.

2. We derive theoretical formula to analyze the performance of our methods. Also, we demonstrate that why a random matrix-based spectral filtering technique proposed by [34] cannot be effectively applied to compromise our methods, even if this filtering technique was designed to challenge the privacy preserving approaches based on random data perturbation.

3. We conduct extensive experiments using the APB benchmark dataset from [35]. Various parameters, such as the *privacy factor* and the *accuracy factor*, have been considered and tested in the experiments. Our experimental results show that both our methods have fulfilled three design goals: security, accuracy, and accessibility.

Chapter 2

Zero-sum method

In this chapter, we introduce our privacy preserving method. The method we proposed is based on **random data distortion**, but it is specifically aimed at solving the *data cube summation problem*. Firstly, we introduce the random data distortion method and analyse why it can not be directly applied to data cubes. Secondly, our privacy preservation methods are proposed. Then the security issues of the method is discussed. Finally,the evaluation factors and the experimental results are presented.

2.1 Random data distortion

2.1.1 Main idea

Privacy preservation based on random data distortion (also known as fix-data perturbation) was developed by J.F.Traub in [21]. This method worked as follows. Assume that the true value of a given attribute (e.g. sales) of an entity k is y_k. Using this method, the result of a range-sum query on y_k will be $T = \sum_{k=1}^{n} x_k$, where $x_k = y_k + e_k$, e_k is a random variable with the expectation $E(e_k) = 0$, the variance $Var(e_k) = \sigma_e^2$, and $\{e_k\}$ are independent.

From statistics literature, two popular scrambling methods, *discretization* and *value perturbation* can be used:

1. *Discretization* – the method often used for hiding individual values. In this method, the values in a block were averaged and the average value was used for every cell in the partition.

2. *Value perturbation* – the basis of this method was to use a value $x_i + z_i$ instead of the true value x_i where z_i was a random value drawn from some distribution. For example, using a uniform distribution between [-α , +α], the mean of the random variable is 0.

The advantages of the random data distortion algorithm are as follows:

1. The method is simple;

2. The distorted data looks very different from the original data and it is almost impossible to accurately estimate the original data;

3. The distorted data can be open to many users without introducing any access restrictions; and

4. Unlike the quasi-cube which needs an additional estimation procedure, all existing commercial OLAP applications can use such distorted data without any change.

However, this method could not be directly applied to data cubes. Because the goal of the method is providing high quality estimates at a "point" level while in our project, what we concern is to provide high quality estimations for aggregations.

2.1.2 Scaling problem

The random data distortion method replaces the actual value x_i by $x_i + \alpha$. The security level achieved by this method depended on the choice of α. If α was a random

value generated in a fixed range, the random data distortion method may encounter the *scaling problem*. For example, if the stock a customer bought is 15,000, adding 8000 to the value would be sufficient to hide the actual value. However, if perturbing 150,000 by 8000, the actual data protection would be considered limited.

An alternative approach was to use *relative value perturbation* instead of absolute value perturbation. Relative value perturbation means that the perturbation is generated from a relative value, say 20%. For instance, if the value of a cell in a data cube is x, the perturbation generated for this cell is within the range $[-|x|20\%, |x|20\%]$.

2.1.3 Query size

By the law of large numbers, the random data distortion method can increase the range query accuracy by applying queries to large query sets. In other words, the error bound of query results depended on the size of the query sets. Hence, the query performance would be degraded dramatically when the query size became small.

Several terminologies will be defined before the privacy preserving method is described.

2.2 Terminologies

Definition 1 *A data cube Ω of d dimension is a d-dimensional array. For each dimension i, the size is n_i, which represents the number of distinct values for that dimension. Thus, the data cube consists of $n_1 \times n_2 \times ... \times n_d$ cells, and each cell can be represented as $\Omega[X_1, X_2, ..., X_d]$ where $0 \leq X_i < n_i$.*

Definition 2 *Given a data cube Ω of d dimensions and the size of each dimension being $n_i (1 \leq i \leq d)$, with d partition factor $b_1, \ldots, b_d$, the data cube can be partitioned into $\prod_{i=1}^{d} (\lceil n_i/b_i \rceil)$ disjoint sub-regions known as blocks or partitions.*

Definition 3 *The input to a range-sum Query can be expressed as $(l_1 : h_1, ..., l_d : h_d)$, where l_i and $h_i (1 \leq i \leq d)$ denote the low and high bound of the range query in ith dimension of the data cube. The range-sum query problem can be formulated as follows:*

$$sum(l_1 : h_1, ..., l_d : h_d) = \sum_{X_1=l_1}^{h_1} \ldots \sum_{X_d=l_d}^{h_d} \Omega[X_1, X_2, ..., X_d]$$

Definition 4 *The size of a range query $Q(l_1 : h_1, ..., l_d : h_d)$ is defined as the number of cells in the query, which can be formulated as:*

$$|Q| = (h_1 - l_1) \times (h_2 - l_2) \times ..... \times (h_d - l_d)$$

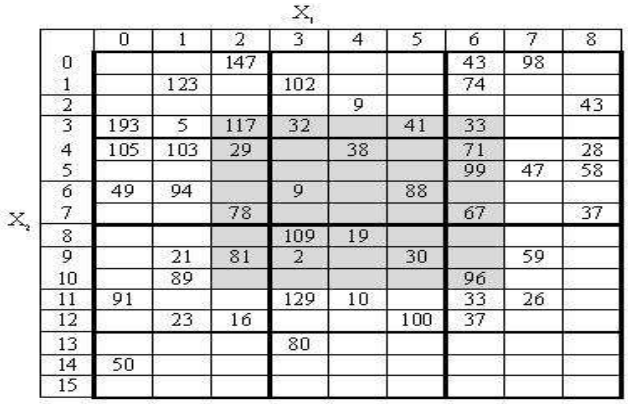
Usually, the data cube is not fully occupied. Some cells are empty, i.e., they contain NULL value because they actually have no corresponding records. The sparsity of a data cube is defined as $\frac{number\ of\ null\ cells}{total\ number\ of\ cells}$.

Generally sparsity ranges from 60% to 90% in the real dataset.

Example 1 *Figure 2.1 shows a simple 2-dimensional data cube. The size of its dimension is 9 × 16. Ω[3, 11] =129, Ω[6, 4] =71. Sparsity = $\frac{95}{9 \times 16}$ = 66%. The partition factors are 3 and 4 for the dimension X_1 and X_2 respectively. The shaded area indicates the query range. The size of the range query is 40.*

2.3 Zero-sum method

As stated in Section 2.1, a value distortion based algorithm has certain attractive features. However, a distortion based method cannot be effectively applied to OLAP



X_2 \ X_1	0	1	2	3	4	5	6	7	8
0			147				43	98	
1		123		102			74		
2					9				43
3	193	5	117	32		41	33		
4	105	103	29		38		71		28
5							99	47	58
6	49	94		9		88			
7			78				67		37
8				109	19				
9		21	81	2		30		59	
10		89					96		
11	91			129	10		33	26	
12		23	16			100	37		
13				80					
14	50								
15									

Figure 2.1: A 2-dimensional data cube

directly. Besides the scaling problem and low accuracy of small queries, the OLAP cube is usually sparse. Simply applying the value distortion, random data distortion will cause the distorted cube's sparsity decreasing dramatically, even to almost 0%. Furthermore, the distortion method applied to OLAP must guarantee a certain degree of accuracy of range-sum queries. These were not considered in random data distortion method. Our solution to this problem is to "adjust" the initially distorted data so that the accuracy of range-sum queries is close to 100%, especially when dealing with small queries.

In this section, we introduce our privacy preservation method for data cubes: the zero-sum method which will be used to perturb the database once in the beginning, and that all OLAP queries would be directly to that perturbed database.

2.3.1 The main idea

First, we start with an initial distortion in each cell. The initially distorted data is then "adjusted" so that all the *marginal sums* of each block are zeroes. These "adjusted" distortions are the final distortions. This adjustment process is called the *zero-sum* method.

For illustration purpose, consider a case of 2-dimensional data cube. The process of *zero-sum* is to make adjustments on each row such that the summation of its (new) distortions are zero. Subsequently, each column's distortion values are also summed up to zero.

There are several ways to enforce *marginal sums* to be zeroes. One of the ways is given as follows: for each row, redistribute the negative sum of the row back to each cell in the row such that the new sum becomes zero. This redistribution can be done uniformly so that each cell receives the same amount of adjustment. After the row adjustments are done, we repeat the same process for column adjustments. This process converts the original distortions to zero-sum form, and a proof is given in Theorem 1.

Theorem 1 *A block (partition) of k-dimension can be converted to zero-sum form with k iterations.*

Proof: Let $d_{i_1,i_2,...,i_k}$ = initial distortion value for cell $\langle i_1, i_2, ..., i_k \rangle$. $1 \leq i_j \leq n_j$, $1 \leq j \leq k$. Let $S_{i_1,i_2,...,i_{k-1},*}$ be the (marginal) sum of distortions at marginal point $(i_1, i_2, ..., i_{k-1}, *)$. Then, we have $S^0_{i_1,i_2,...,i_{k-1},*} = \sum_{i_k=1}^{n_k} d_{i_1,i_2,...,i_{k-1},i_k}$

Similarly,

$$S^0_{i_1,i_2,...,i_{k-2},*,*} = \sum_{i_{k-1}=1}^{n_{k-1}} \sum_{i_k=1}^{n_k} d_{i_1,i_2,...,i_{k-2},i_{k-1},i_k}$$
$$= \sum_{i_{k-1}=1}^{n_{k-1}} S^0_{i_1,i_2,...,i_{k-1},*}$$

is defined as the (marginal) sum of distortions at marginal point $(i_1, i_2, ..., i_{k-2}, *, *)$. In the same way, we can define the (marginal) sum of distortions for all other marginal points.

Let $\Delta_{i_1,i_2,...,i_{k-1},*} = -\frac{S^0_{i_1,i_2,...,i_{k-1},*}}{n_k}$

In the first iteration, we have

$d_{i_1,i_2,...,i_k} \leftarrow d_{i_1,i_2,...,i_k} + \Delta_{i_1,i_2,...,i_{k-1},*}$

Now, the value $S^1_{i_1,i_2,...,i_{k-1},*}$ for marginal point $(i_1, i_2, ..., i_{k-1}, *)$ after the first iteration becomes zero. Similarly, $S^1_{i_1,i_2,...,i_{k-2},*,*}$ also becomes zero, and so on, for the sum of distortions at all other marginal points.

Repeat the process, and after the second iteration, the value $S^2_{i_1,i_2,...,i_{k-2},*,i_k}$ becomes zero.

However, the value $S^2_{i_1,i_2,...,i_{k-1},*}$ still remains zero. This is because

$$\begin{aligned} S^2_{i_1,i_2,...,i_{k-1},*} &= S^1_{i_1,i_2,...,i_{k-1},*} + \frac{\sum_{i_k=1}^{n_k} S^1_{i_1,i_2,...,*,i_k}}{n_{k-1}} \\ &= S^1_{i_1,i_2,...,i_{k-1},*} + \frac{S^1_{i_1,i_2,...,i_{k-2},*,*}}{n_{k-1}} \\ &= 0 \end{aligned}$$

Similarly, after the third iteration, the values of $S^3_{i_1,i_2,...,i_{k-1},*}$, $S^3_{i_1,i_2,...,i_{k-2},*,i_k}$, and $S^3_{i_1,i_2,...,i_{k-3},*,i_{k-1},i_k}$ are all zeroes. This can be seen from the following:

For the value $S^3_{i_1,i_2,...,i_{k-1},*}$ we have

$$\begin{aligned} S^3_{i_1,i_2,...,i_{k-1},*} &= S^2_{i_1,i_2,...,i_{k-1},*} + \frac{S^2_{i_1,i_2,...,i_{k-3},*,i_{k-1},*}}{n_{k-2}} \\ &= 0 \end{aligned}$$

For the value $S^3_{i_1,i_2,...,i_{k-2},*,i_k}$ we have

$$\begin{aligned} S^3_{i_1,i_2,...,i_{k-2},*,i_k} &= S^2_{i_1,i_2,...,i_{k-2},*,i_k} + \frac{S^2_{i_1,i_2,...,i_{k-3},*,*,i_k}}{n_{k-2}} \\ &= 0 \end{aligned}$$

And the same is true for all subsequent iterations. □

6	-4	4	6	-1	11
-5	-6	7	-7	-4	-15
-7	-1	-3	5	9	3
8	5	-8	-4	-3	-2
-5	-2	4	3	2	2
-3	3	-7	3	-2	-6
6	-4	6	-5	-3	0
0	-9	3	1	-2	-7

(a) Original distortions

3.8	-6.2	1.8	3.8	-3.2	0
-2	-3	10	-4	-1	0
-7.6	-1.6	-3.6	4.4	8.4	0
8.4	5.4	-7.6	-3.6	-2.6	0
-5.4	-2.4	3.6	2.6	1.6	0
-1.8	4.2	-5.8	4.2	-0.8	0
6	-4	6	-5	-3	0
1.4	-7.6	4.4	2.4	-0.6	0

(b) Distortions adjusted by row

3.6	-5.1	1.2	3.45	-3.1	0
-2.2	-1.9	9.4	-4.3	-0.9	0
-7.8	-0.5	-4.2	4.1	8.48	0
8.2	6.48	-8.2	-3.9	-2.5	0
-5.6	-1.3	3	2.3	1.7	0
-2	5.3	-6.4	3.9	-0.7	0
5.8	-2.9	5.4	-5.3	-2.9	0
0	0	0	0	0	0

(c) Distortions adjusted by column

Figure 2.2: An example of using the iterative zero-sum method

Theorem 1 means that for a k-dimensional block, in ith-iteration, all marginal sums of dimension $i(1 \leq i \leq k)$ can be adjusted to zeroes while all marginal sums of dimension $j(1 \leq j < i)$ remain zeroes. Thus after k-iteration, all marginal sums of the k-dimensional block are zeroes.

Example 2 *An example of zero-sum iterations on a 2-dimensional data cube is given in Figure 2.2. Suppose that there is an original block with 7 rows and 5 columns. Figure 2.2(a) shows the original distortions of the original block. Each cell contains a randomly generated number within [-9,9]. The first iteration is to adjust the sum of each row to be zero. Each row has 5 cells and the sum of the first row is 11. The sum of the first row becomes zero by adding* $-(11/5) = -2.2$ *to each cell of the first row. Then we adjust all other rows in the same manner. After the first iteration, as shown in Figure 2.2(b), the sum of each row is zero.*

Next consider the sum of each column. Similarly, as shown in Figure 2.2(b), the sum of the first column is 1.4. The sum of the first column is adjusted to be zero by adding $-(1.4/7) = -0.2$ *to each of the seven cells in this column.*

The cell values after the second iteration are shown in Figure 2.2(c). It can be seen that the sum of each column is zero, and the sum of each row also remains as zero. The adjusted distortions in Figure 2.2(c) is in zero-sum form. If we add the adjusted distortions to the corresponding cells in the original block, the value of all the cells in the original block will be changed but all marginal sums will remain the same. Therefore after distortion, a certain degree of the accuracy of the range-sum query can be guaranteed to some extent.

Alternatively, using formula 2.1 below, a block can be converted to zero-sum form by one iteration.

$$\begin{aligned} d_{i_1,i_2,\ldots,i_k} \leftarrow d_{i_1,i_2,\ldots,i_k} &- \frac{S_{i_1,\ldots,i_{k-1},*}}{n_k} - \frac{S_{i_1,\ldots,i_{k-2},*,i_k}}{n_{k-1}} - \ldots + \frac{S_{i_1,\ldots,i_{k-2},*,*}}{n_k n_{k-1}} \\ &+ \frac{S_{i_1,\ldots,i_{k-3},*,i_{k-1},*}}{n_k n_{k-2}} + \frac{S_{i_1,\ldots,i_{k-3},*,*,i_k}}{n_{k-1} n_{k-1}} + \ldots - \ldots \\ &+ (-1)^k \frac{S_{*,*,\ldots,*}}{n_k n_{k-1} \ldots n_2 n_1} \end{aligned} \tag{2.1}$$

Example 3 *The same example in Figure 2.2 can be converted to zero-sum form by the 1-iteration method as shown in Figure 2.3. In Figure 2.3(a), the original distortion value* $d_{11} = 6$ *is converted to* $d_{11} - \frac{S_{1,*}}{n_2} - \frac{S_{*,1}}{n_1} + \frac{S_{*,*}}{n_1 n_2} = 6 - \frac{11}{5} - \frac{0}{7} + \frac{-7}{35} = 3.6$ *The result in Figure 2.3(b) is the same as Figure 2.2(c).*

If the integer format is desired, then distortions can be reinstalled by rounding the numbers. First round all the numbers except boundary cells, shown in Figure 2.3(c). Then the boundary values are given to make the marginal sums remain as zeroes. Figure 2.3(d) shows the final result.

The *time complexity* for k-iterations is $k \times n_1 \times n_2 \times \ldots \times n_k$. The time complexity for 1-iteration is $2^k \times n_1 \times n_2 \times \ldots \times n_k$.

6	-4	4	6	-1	11
-5	-6	7	-7	-4	-15
-7	-1	-3	5	9	3
8	5	-8	-4	-3	-2
-5	-2	4	3	2	2
-3	3	-7	3	-2	-6
6	-4	6	-5	-3	0
0	-9	3	1	-2	-7

(a) Original distortions

3.6	-5.1	1.2	3.45	-3.1	0
-2.2	-1.9	9.4	-4.3	-0.9	0
-7.8	-0.5	-4.2	4.1	8.48	0
8.2	6.48	-8.2	-3.9	-2.5	0
-5.6	-1.3	3	2.3	1.7	0
-2	5.3	-6.4	3.9	-0.7	0
5.8	-2.9	5.4	-5.3	-2.9	0
0	0	0	0	0	0

(b) Adjusted distortions modification

4	-5	1	3	3
-2	-2	9	-4	1
-8	-1	-4	4	-9
8	6	-8	-4	2
-6	-1	3	2	-2
-2	5	-6	4	1
-6	2	-5	5	

(c) Round to integer

4	-5	1	3	-3	0
-2	-2	9	-4	-1	0
-8	-1	-4	4	9	0
8	6	-8	-4	-2	0
-6	-1	3	2	2	0
-2	5	-6	4	-1	0
6	-2	5	-5	-4	0
0	0	0	0	0	0

(d) Result in integer format

Figure 2.3: Example of converting a block to zero-sum form in 1-iteration

In reality, OLAP systems often use existing languages such as SQL to express the OLAP operations. From this angle, the slice and dice, and the roll-up/drill-down operations can be viewed as range queries.

Figure 2.4 illustrates how the zero-sum method preserves the confidential information in each data cell while still being able to provide an accurate estimation of the original summation values for range queries. The data set is a 2-dimensional data cube with 9 blocks. The gray rectangle in Figure 2.4 is a range query. This range query covers 9 blocks: $A_1, A_2, ..., A_9$ and it fully covers A_5 only. According to the zero-sum method, the cells in each block have been "adjusted", so the marginal sums of a distorted block is equal to that of corresponding original block. Let S be the sum of the range query over original data, and S' be the sum of distorted data. Let s_i be the sum of the gray area in the block A_i before distortion, and s'_i be the sum of the gray area in the block A_i after distortion.

$$S = s_1 + s_2 + ... + s_9$$

$$S' = s'_1 + s'_2 + ... + s'_9$$

$$S' - S = (s'_1 - s_1) + (s'_3 - s_3) + (s'_7 - s_7) + (s'_9 - s_9).$$

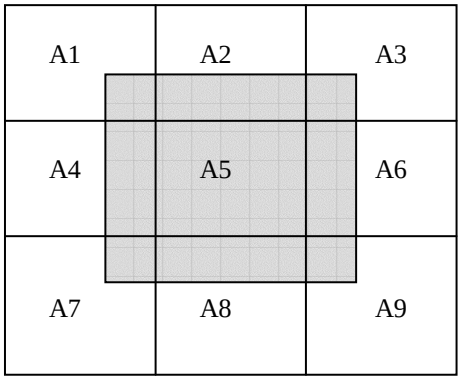


Figure 2.4: Data cube, blocks and query range

It can be seen that $S' - S$ is a small fraction of S. That's why the zero-sum method can help to provide an accurate estimation for range-sum queries.

2.3.2 A special case

A special case is that all cells are initially distorted to a constant value. For example, when *discretization* is used as the initial distortion, the values in a block are averaged and the average value is used for every cell in the block. In this special case, the (adjusted) final distorted values will be in a special format. For instance, in the 2-dimensional case, the new value y_{ij} at cell *(i,j)* becomes

$$y_{ij} = x_{ij} + z_{ij} = \frac{\sum_{j=1}^{n} x_{ij}}{n} + \frac{\sum_{i=1}^{m} x_{ij}}{m} - \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}}{mn}$$

In general, in the case of k-dimensional data cube, we have

$$\begin{aligned} y_{i_1,i_2,\ldots,i_k} = & \frac{\sum_{i_k=1}^{n_k} x_{i_1,\ldots,i_{k-1},i_k}}{n_k} + \frac{\sum_{i_{k-1}=1}^{n_{k-1}} x_{i_1,\ldots,i_{k-1},i_k}}{n_{k-1}} + \ldots \\ & - \frac{\sum_{i_{k-1}=1}^{n_{k-1}}\sum_{i_k=1}^{n_k} x_{i_1,\ldots,i_{k-1},i_k}}{n_{k-1}n_k} - \frac{\sum_{i_{k-2}=1}^{n_{k-2}}\sum_{i_k=1}^{n_k} x_{i_1,\ldots,i_{k-1},i_k}}{n_{k-2}n_k} \\ & - \frac{\sum_{i_{k-2}=1}^{n_{k-2}}\sum_{i_{k-1}=1}^{n_{k-1}} x_{i_1,\ldots,i_{k-1},i_k}}{n_{k-2}n_k} - \ldots + \ldots - \ldots \\ & - (-1)^k \frac{\sum_{i_1=1}^{n_1}\cdots\sum_{i_k=1}^{n_k} xi_1, i_2, \ldots, i_n}{n_k n_{k-1}\ldots n_2 n_1} \end{aligned} \tag{2.2}$$

2.3.3 Error bound analysis

In this section, we present theoretical analysis of the zero-sum method. We derive a bound on distortions that is useful when explaining in probabilistic terms. A generalization of the commonly known *Chernoff's* bounds ([36]) is used to derive the bound.

Theorem 2 *let* X_i, $1 \leq i \leq n$, *be mutually independent random variables with all* $E[X_i] = 0$ *and all* $|X_i| < \alpha_i$. *Let* $S_n = X_1 + ... + X_i$. *For* $a > 0$, *then*

$$Pr[S_n > a] < e^{-a^2/2\sum \alpha_i^2}$$

Proof: Let $\lambda = a/\sum \alpha_i^2$ and $h_i(x) = \frac{e^{\lambda\alpha_i}+e^{-\lambda\alpha_i}}{2} + \frac{e^{\lambda\alpha_i}-e^{-\lambda\alpha_i}}{2\alpha_i}x$

For $x \in [-\alpha_i, \alpha_i]$, $e^{\lambda x} \leq h_i(x)$. ($y = h_i(x)$ is a chord passing through the points $x = -\alpha_i$ and $x = +\alpha_i$ of a convex curve $y = e^{\lambda x}$.) Thus

$$\begin{aligned} E[e^{\lambda X_i} \leq E[h_i(X_i)] &= h_i(E[X_i]) \\ &= h_i(0) \\ &= \frac{e^{\lambda\alpha_i} + e^{-\lambda\alpha_i}}{2} \\ &= cosh(\lambda\alpha_i) \end{aligned}$$

The inequality below is valid for all $\lambda > 0$. (The inequality can be shown by comparing the Taylor series of the two functions $e^{\lambda\alpha_i}$ and $e^{-\lambda\alpha_i}$ term-wise.)

$$cosh(\lambda\alpha_i) < e^{\lambda^2\alpha_i^2/2}$$

We have $e^{\lambda S_n} = \prod_{i=1}^{n} e^{\lambda X_i}$

Since the X_i are mutually independent, so are $e^{\lambda X_i}$, and

$$\begin{aligned} E[e^{\lambda S_n}] &= \prod_{i=1}^{n} E[e^{\lambda X_i}] \\ &= \prod_{i=1}^{n} cosh(\lambda \alpha_i) \\ &\leq e^{\lambda^2 \sum \alpha_i^2/2} \end{aligned}$$

Note that $S_n > a$ if and only if $e^{\lambda S} > e^{\lambda a}$. Apply the following Markov's inequality

$$Pr[Y > aE[Y]] < \frac{1}{a}$$

we get

$$\begin{aligned} Pr[S_n > a] &= Pr[e^{\lambda S_n} > e^{\lambda a}] \\ &< E[e^{\lambda S_n}]/e^{\lambda a} \\ &\leq e^{\lambda^2 \sum \alpha_i^2/2 - \lambda a} \end{aligned}$$

Since $\lambda = a/\sum {\alpha_i}^2$, the inequality becomes

$$Pr[S_n > a] < e^{-a^2/2\sum \alpha_i^2}$$

Hence, the claim holds. □

From the above Theorem 2, we can derive the following two corollaries.

Corollary 1 *Under the same conditions as Theorem 2, we have* $Pr[|S_n| > a] < 2e^{-a^2/2\sum \alpha_i^2}$

For a range query Q, x_i denotes the final distortion added to ith cell of Q, S(Q) denotes the summation of query Q before perturbation while S'(Q) denotes the summation of the same query after perturbation. Then $S_n = S(Q) - S'(Q)$ is bounded by Theorem 2.

2.3.4 Handling null cells

In reality, the sparsity of data cubes is usually between 60% and 90%. Therefore, more than half of data cells are null. If the zero-sum method is applied to perturb a data cube without considering the presence of null cells, the sparisty of the perturbed data cube will decrease to 0%. This will lead to high storage cost.

Example 4 *Figure 2.5 shows how the null cells became non-null cells after distorted with zero-sum method.*

Figure 2.5(a) shows the original distortions of the original block. The first iteration is to adjust the sum of each row to zero. The result is shown in figure 2.5(b). It can be seen that the sum of each row is zero but the two null cells became non-null cells.

For the second iteration, e.g. adjusting the sum of each column to zero. The result is illustrated in figure 2.5(c). All marginal sum of distortions are zero but all null cells are now non-null cells.

8	null	7	15
null	5	8	13
-4	-2	9	3
4	3	24	

(a) Original distortions

3	-5	2	0
-4.33	0.67	3.67	0
-5	-3	8	0
-6.33	-7.33	13.67	

(b) Distortions adjusted by row

5.11	-2.56	-2.56	0
-2.22	3.11	-0.89	0
-2.89	-0.56	3.44	0
0	0	0	

(c) Distortions adjusted by column

Figure 2.5: An example of zero-sum without considering null cells

So, in order to keep the sparsity of original data cubes, null cells must be taken into the consideration. A simple way to handle the null cells is to take the null cells as

uncounted. As a result, all null cells would remain null, but the adjusted distortion can not be kept in strict zero-sum form. However, our experiment results show that such non-strict zero-sum form did not affect query accuracy much. Example 5 illustrates how we keep the null cells to be null when adjusting the original distortions.

Example 5 *For the same block shown in figure 2.5(a). At the first iteration, since the first row contains one null cell, the number of cells of the first row would be two. The sum of the first row is 15. In order to adjust it to zero, each of two non-null cells would be added* $-(15/2) = -7.5$. *Similarly, the number of cells of the second row is also taken as two. But the number of cells of the third row is three because of no null cell in the row. After the first iteration, as shown in figure 2.6(a), the sum of each row is zero.*

The same procedure is applied to the second iteration. Figure 2.6(b) shows the result of the second iteration. It can be seen that although the sum of each column is zero, the sum of each row is changed to non-zero.

0.5	null	-0.5	0
null	-1.5	1.5	0
-5	-3	8	0
-4.5	-4.5	9	

(a) Distortions adjusted by row

2.75	null	-3.5	-0.75
null	0.75	-1.5	-0.75
-2.75	-0.75	5	1.5
0	0	0	

(b) Distortions adjusted by column

Figure 2.6: An example of zero-sum handling null cells

2.4 Security and other issues

In this section, we discuss some issues of the zero-sum method:

Security: Since the (original) cubes are not allowed for probing, it's not possible for snoopers to improve their estimation of the value of a field in a record by repeatedly placing queries ([14]).

It was shown in [37] that it was possible to fully recover original distribution from non-overlapping, contiguous partial sums. However, it required the original distribution to be smooth enough. (That is there will only be a small difference between successive elements of the vector.) This "smooth" assumption is not true in general for data cubes.

Research work on estimating attribute distributions from partial information can be found in [38], or on approximating queries on sub-cubes from higher-level aggregations can be found in [7]. However, these works did not deal with information that has been deliberately distorted. Furthermore their estimations required some additional statistical information of actual data.

Another security attack can come from the known marginal information. For example, in the case of 2-dimensional data cubes, the correct values of the sums of rows and the sums of columns are known. However, this known data is relatively much smaller than the unknown information. In the case of 2-dimensional data cubes, for example, it is impossible to solve a system of $(m + n - 1)$ equations using $m \times n$ variables, where m is the number of rows and n is the number of columns.

Recently, [34] proposed a random matrix-based spectral filtering technique to challenge the privacy preserving approaches based on random data perturbation. Although random data distortion is used in our paper, this filtering technique cannot be effectively applied to compromise our approach for the following three reasons. Firstly, the effectiveness of this filtering technique was based on the assumption that the eigenvectors of the covariance matrix of the perturbed data were orthogonal to

the eigenvectors of the covariance matrix of the original data. Consider that, in our approach, different blocks in a data cube can be distorted by random data from different distributions, such as normal distribution, uniform distribution or Gaussian distribution etc., with different ranges. So the eigenvectors of the covariance matrix of the perturbed data are not orthogonal to the eigenvectors of the covariance matrix of the original data. In other words, one of the major assumptions in this filtering approach is invalid in our paper. Secondly, if there was no prior knowledge of perturbed data distributions, which is true in our approach, this filtering approach needed to estimate the variance of the perturbed data. Even if this variance can be correctly estimated, the process for estimating the variance adds complexity and increases computation cost significantly. Finally, the filtering approach required to compute the eigenvalues of a covariance matrix (computation is O(M^3) where M is the matrix dimension [39]), this computation cost for a large amount data, such as a data cube, can be intractable.

Pre-computing: The paper presented by [4] introduced the idea of pre-computing multidimensional prefix-sums of a data cube for speeding up range-sum query. Prefix-sum means, for any cell at a position $\langle d_1, d_2, ..., d_n \rangle$ in a data cube, summing up all the values in the range starting from the position $\langle 0, ..., 0 \rangle$ to $\langle d_1, d_2, ..., d_n \rangle$. The resulting value of the summation was then stored in the cell of $\langle d_1, d_2, ..., d_n \rangle$ of the auxiliary data structure. Since this summation looked like a prefix of the summation of the whole data cube, it was given the name *prefix-sums*. By pre-computing as many prefix-sums as the number of elements in the original data cube, any range-sum query could be answered by accessing and combining 2^d appropriate prefix-sums, where d was the number of dimensions for which ranges have been specified in the query. Pre-computing had no effect on the data security but only affects the tradeoff between response speed and storage space.

Size of block: Using zero-sum method, a data cube is divided into blocks and each dimension should have at least two values. The boundary of a block, if matched with a dimensional hierarchy, will guarantee that the roll-up operation is completely correct.

Sparsity: Sparse cubes are cubes in which many cells are empty. An empty cell means that there is no valid value in the cell. These cells can be treated as either zeroes, or as missing values that are not counted. If they are treated as zeroes, after scrambling, a zero cell may become non-zero. This decreases the sparsity of a block. Also, counting empty cells as zeroes will affect the calculation on average. However, if empty cell are treated as uncounted, formula 2.1 cannot guarantee to produce zero-sum forms. This is because the number of counted cells in each row or column can be (very) different. So theorem 1 is no longer valid. However, the experimental results show that such a non-strict zero-sum form does not affect the accuracy much.
A special case is that a row (or column) contains no more than one valid cell. Our zero-sum method simply leaves this row (or column) unadjusted.

Absolute Data Distortion vs. Relative Data Distortion: Absolute data distortion is to distort original data by an absolute value, whereas relative data distortion is to distort data by a relative value, say 20%. The absolute distortion suffers in terms of scale. For example, perturbing a salary of $15,000 by 3,000 would preserve the confidentiality of the data while at the same time perturbing a salary of $150,000 by 3,000 would be considered a compromise. Perturbing the data in a relative distortion range is an alternative way to overcome the scale problem.

2.5 Performance measurement

To evaluate the performance of the zero-sum method, two measures *privacy* and *accuracy* are used.

2.5.1 Measure of privacy

The measure of privacy indicates how closely the original value can be estimated. Original distortions are usually generated by using a density function $f(z)$ in an interval of $[0, \alpha]$, and then assign positive or negative values with equal probability to it. Therefore, the expectation of distortion variable, z, is $E(z) = 0$. The expectation of variable $|z|$ is

$$E(|z|) = \int_{-\alpha}^{\alpha} |z| \frac{f(z)}{2} dz = \int_{0}^{\alpha} z f(z) dz$$

The work by Agrawal and Srikant in [24] used a simple measure in terms of the amount of privacy, which was defined as follows: assume the original distortion was uniformly distributed in an interval of $[0, \alpha]$, then the expectation was intuitively considered as the amount of privacy.

We generalize the privacy measure in [24] to all types of distributions based on the mean value (expectation): if the (original) distortion distribution has density function $f(z)$ in $[0, \alpha]$, then the amount of privacy is defined as $2 \times \int_0^\alpha z f(z) dz = 2E(|z|)$.

A more sophisticated privacy measure based on differential entropy is given by [27]: if the (original) distortion distribution has density function $f(z)$ in an interval of $[0, \alpha]$, then the amount of privacy is $2^{-\int_0^\alpha f(z) log_2 f(z) dz}$.

However, in this chapter, we are more interested in "what is the expected amount of privacy of the adjusted distortions". Since we do not know the density function of the adjusted distortions, we can only estimate the amount of privacy at cell i by

using the difference between x_i (the true original value) and y_i (the adjusted distorted value). Based on the spirit of expectation and the law of large numbers, we will give the definition of a privacy factor below and use it to define the privacy amount of the adjusted distortions.

Privacy factor F_p:

$$F_p = \frac{1}{N}\sum_{i=1}^{N}|y_i - x_i| \tag{2.3}$$

In formula 2.3, the value F_p is the *average* amount of (adjusted) distortions over a block. The value $2 \times F_p$ is considered as the amount of privacy of the adjusted distortions. It is interesting to compare the value $2 \times F_p$ with the amount of privacy of the original distortions. The privacy amount calculated in formula 2.3 does not take into account the value of the original data. To quantify privacy accurately, we need a method which takes such information into account. A modified version of formula 2.3 is given below:

$$F_c = \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - x_i|}{|x_i|} \tag{2.4}$$

The value F_c is the average amount of distortions (relative to x-value) over a block. The value $2 \times F_c$ is considered as the amount of *conditional privacy* of the adjusted distortions. Accordingly, the *conditional* privacy of the original distortions is also modified by a factor x_i at cell i. That is, a cell with original value x will generate distortions that are (uniformly) distributed over an interval of $[0, \alpha x]$.

Generally speaking, the measurement of privacy is dependent on the expectation, $E(|X|)$, of the distortion distribution.

2.5.2 Measure of accuracy

The difference between the sum of the perturbed values and the actual values over a range query Q is referred to as the *accuracy loss* of Q. Let *true_sum* be the sum of all actual values of cells in query Q, and *answer* be the sum of all perturbed values. Formula (2.5) defines the *relative accuracy loss* of Q, d_Q.

$$d_Q = |\frac{answer \ - \ true\_sum}{true\_sum}| \tag{2.5}$$

We define the measure of accuracy as formula (2.6) shows.

Accuracy factor $F_{a,Q}$:

$$F_{a,Q} = \frac{1}{1 + d_Q} \tag{2.6}$$

Note that the accuracy factor $F_{a,Q}$ lies between 0 and 1 (inclusive). If the accuracy factor is close to one, the range query has a high query accuracy. If the accuracy factor is close to zero, the query has a low query accuracy. When the estimated *answer* equals to the *true_sum*, accuracy factor is 1 (100%).

2.6 Experiment result

2.6.1 Experiment setup

Experimental Data Sets. The experimental data sets are generated using the APB Benchmark program from [35]. These data sets had four dimensions: customer, product, channel, and time. The size of each dimension was 900 (customer), 9000 (product), 9 (channel) and 17 (time) respectively. The measure of interest was dollar, with range [0, 699] and data type *short*. Cells with -1 dollar value were empty cells and were treated as missing or uncounted. The overall sparsity of the data

cube was 80%. Hence, the file size of the data cube was 900*9000*9*17*(1-0.8)*2 = 495,720,000 bytes = 0.49 GB. The original size of data file generated by APB Benchmark was approximately 15G (ASCII text). Each record in this file was read to fill the corresponding cells in the Data Cube.

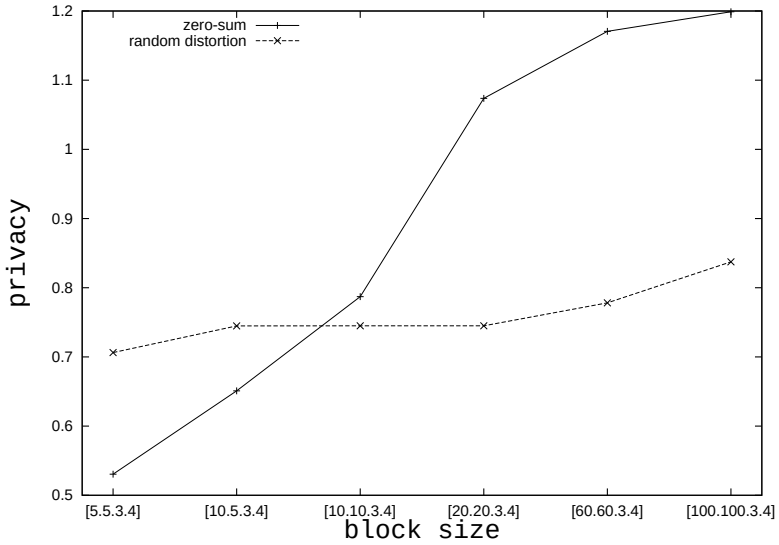
Distortion range. Relative distortion range was applied to generate initial distortions. For example, given a relative distortion range [0, 50%], and a cell value α, the initial distortion of this cell is randomly generated within range [-50%α, 50%α].

Experimental Platform. Our experiments were performed on a PC with a Pentium III 733 MHz, 40G hard disk, and 256 MBytes of memory.

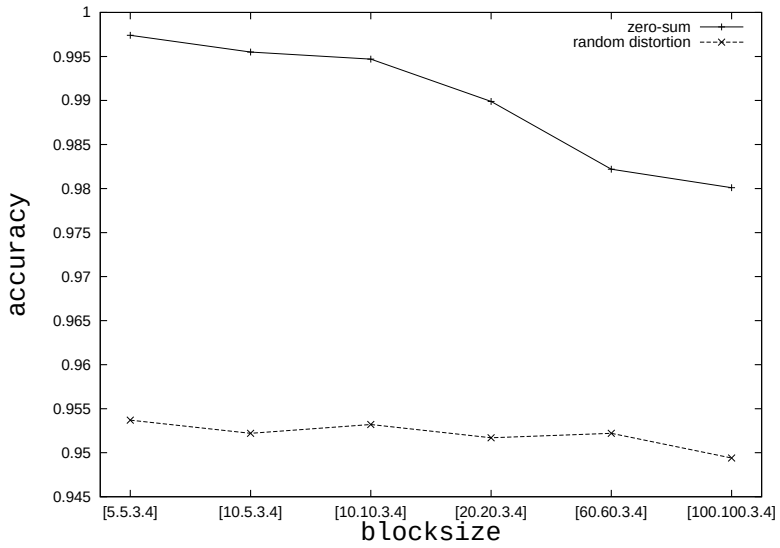
2.6.2 The effect of block sizes

To test the effect of block sizes, we arranged the distortion range to be [50%, 100%]. In this experiment, 600 range-sum queries with query size in [200, 1000] were generated. Figure 2.7(a) shows the obtained privacy values for zero-sum method and random data distortion method. Although the privacy values of zero-sum were smaller than that of random distortion when the block size was small, the overall privacy of zero-sum was better than that of random distortion. Figure 2.7(b) shows that the accuracy values of zero-sum were significantly and consistently better than the accuracy values of random distortion regardless of the block size.

In order to see the relationship between privacy and accuracy, we put the curves representing the performance of zero-sum method in Figure 2.7(a) and Figure 2.7(a) into one figure, Figure 2.8. It shows the privacy and accuracy values with the change of block size. The privacy of zero-sum increased as the block size increased. However, the accuracy decreased with the increase of the block size. In other words, there was a trade-off between privacy and accuracy.



(a) Privacy of different block size



(b) Accuracy of different block size

Figure 2.7: The effect of different block size

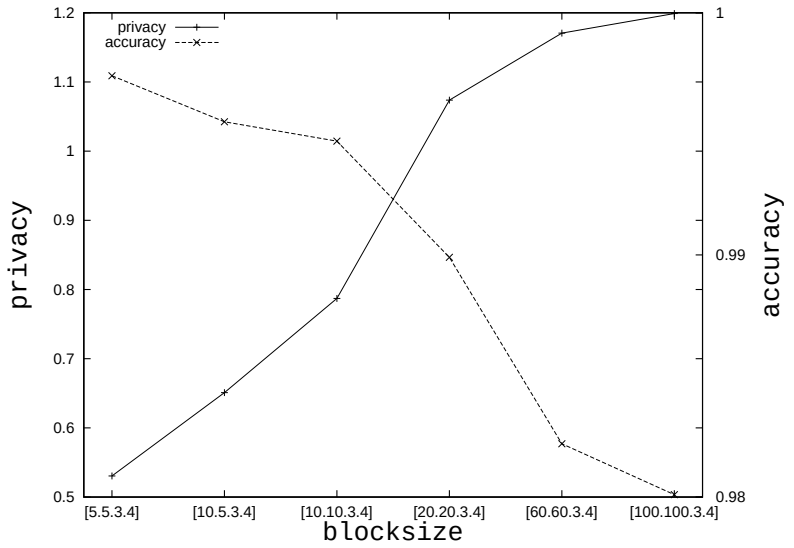


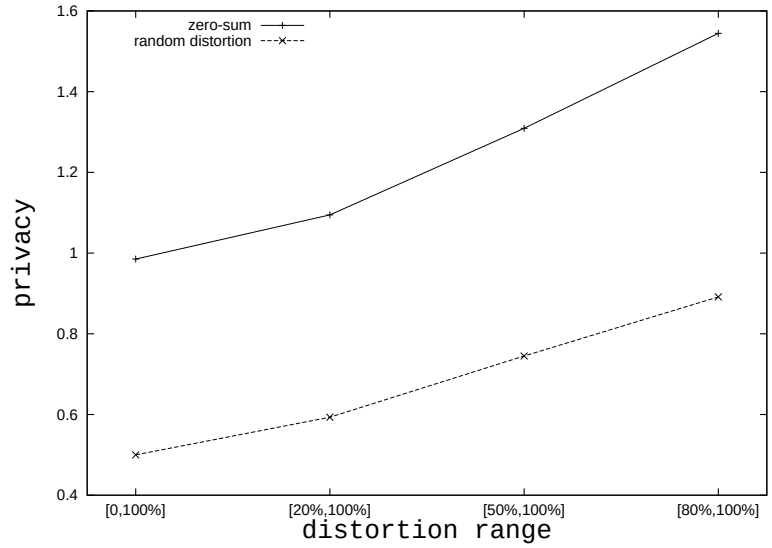
Figure 2.8: Relationship between privacy and accuracy

2.6.3 The effect of distortion range

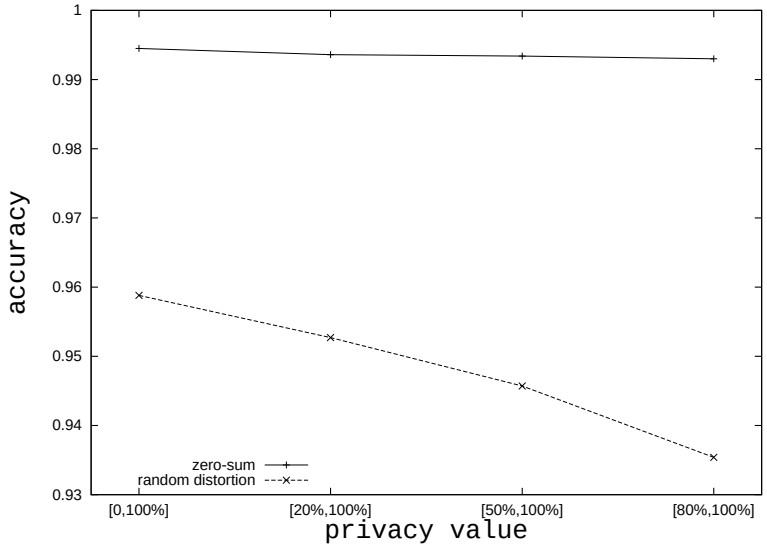
In the experiment of examining the effect of distortion range, we arranged the block size to be 20*20*3*4 and the cube sparsity to be 60%. We applied 600 range-sum queries with query sizes between 200 and 1000. We then examined the average privacy, as well as the accuracy of these 600 queries, with the change of distortion ranges. Figure 2.9(a) shows the privacy performance of the zero-sum and random distortions when changing the lower bound of distortion range. In the figure, it was not surprising to see a trend that the privacy values of both methods increased with the increase of the lower bound of relative distortion ranges, since higher relative distortion could bring better privacy preservation. Also, we observed that the obtained privacy of zero-sum was consistently better than that of random distortion.

Figure 2.9(b) shows the accuracy performance of the zero-sum and random distortions when changing the lower bound of distortion range. The accuracy of zero-sum was significantly and consistently higher than that of random distortion. Furthermore, the accuracy of random distortion decreased with the increase of the lower

bound of distortion range. In contrast, changing distortion ranges did not affect significantly the accuracy of zero-sum.



(a) Privacy



(b) Accuracy

Figure 2.9: The effect of different distortion range

2.6.4 The effect of cube sparsity

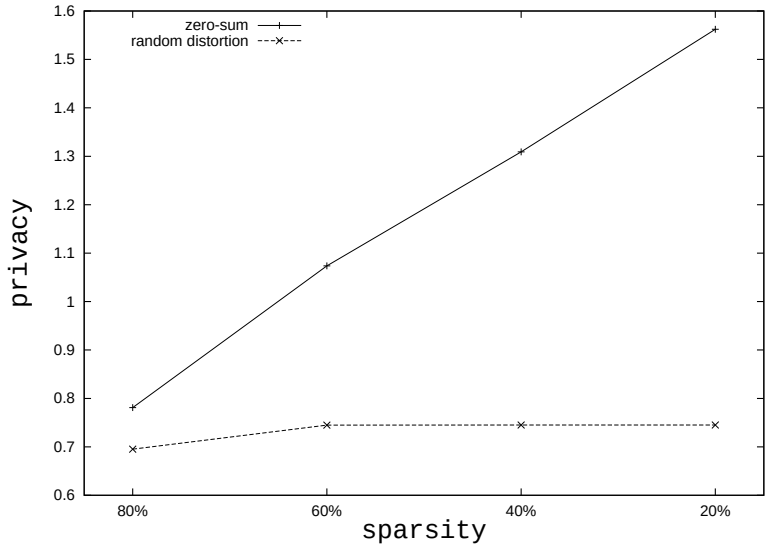
To test the effect of cube sparsity, we arranged the block size to be 20*20*3*4 and the distortion range to be [50%, 100%]. In this experiment, we applied 600 range-sum queries with the query size of [200, 1000]. Figure 2.10(a) shows the obtained privacy values of the two methods. As shown in the figure, the privacy of zero-sum was consistently better than that of random distortion. In addition, we observed that the privacy of zero-sum increased with the decrease of sparsity.

Figure 2.10(b) shows the accuracy of zero-sum and random distortion with the change of the cube sparsity. As shown, the accuracy of zero-sum was significantly better than that of random distortion in all ranges of the cube sparsity. In addition, the accuracy of random distortion slightly increased with the decrease of sparsity while the accuracy of zero-sum was almost not affected by the different cube sparsity.

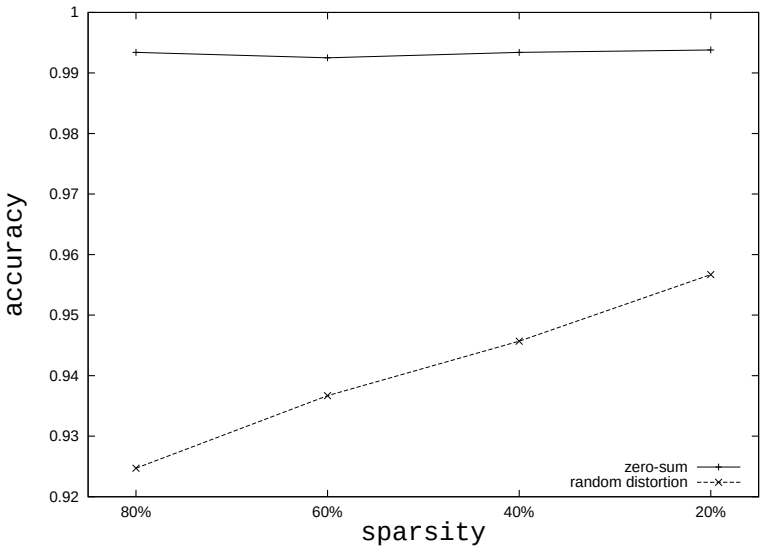
In summary, the more sparse the cube was, the smaller the privacy would be. But with the zero-sum method, the cube sparsity did not affect greatly the accuracy of range-sum queries.

2.6.5 The effect of query sizes

Here, we tested the effect of query sizes using a fixed block size as 20*20*3*2. More specifically, we examined the performance of adjusted distortion as well as original distortion at the same privacy level. The privacy of random distortion was 0.89 and the privacy of zero-sum was 0.99 (It is almost impossible to get exactly the same privacy value for the two different method.). The size of queries ranged from (<50), (50, 200), (200, 500) to (500, 1000), and we generated 200 range-sum queries for each category. The experimental results are shown in Figure 2.11. For the original distortion, when the query size decreased, the accuracy decreased dramatically. When



(a) Privacy of different sparsity



(b) Accuracy of different sparsity

Figure 2.10: The effect of different cube sparsity

the size of range query was smaller than 50, the accuracy of random distortion was below 80% which was not acceptable. While for zero-sum, although the accuracy also decreased with the decrease of query size, the accuracy value was always above 96%, even for very small range queries.

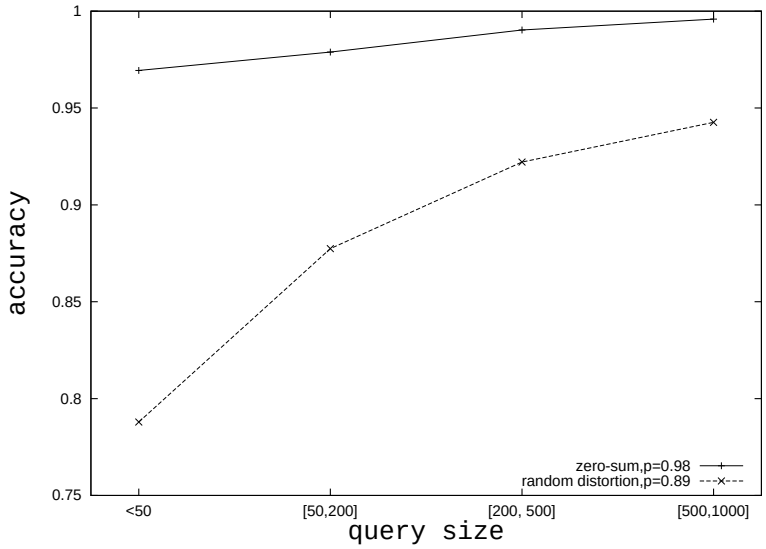


Figure 2.11: The accuracy of different query size

2.6.6 Summary of experimental results

Several major experimental results are summarized as follows.

1. The accuracy of adjusted distortions was significantly and consistently better than the accuracy of original distortions. This is due to the fact that the zero-sum method constrains the marginal sums of distortions to be zero. In section 2.3.1, we have shown in Figure 2.4 that only the partially covered blocks may yield the inaccuracy of range-sum query.

2. Based on our experimental results, using relative data distortion, our zero-sum method obtained better privacy in many cases than the privacy of original distortions. This is because each marginal sum of the original distortion is

redistributed back to each cell in the block. If the same amount of adjustment is applied to two different valued cells, the cell with smaller value has more significant change in comparison with the change in the cell with larger value. Therefore, when computing relative privacy, the gains by smaller valued cells will exceed the losses by larger valued cells. Thus, in general, our method can achieve better results.

3. The three parameters which affected the performance of zero-sum method were block size, distortion range and cube sparsity. The cube sparsity can not be adjusted. The parameter *block size* affected the performance of zero-sum method much more than the parameter *distortion range* did. When block size was small, the accuracy was high but the privacy was low. With larger block size, the zero-sum methods could obtain better privacy but the accuracy decreased. In other words, there was a trade-off between privacy and accuracy with different block size.

Chapter 3

Cubic-wise balance method

3.1 Motivations

Chapter 2 introduced a simple and effective method, zero-sum method, for data cube privacy preservation. With the idea of constraining all marginal sums of each data block to be zero, the zero-sum method could accurately estimate the answer of range-sum queries although the raw data has been intentionally perturbed.

To apply the zero-sum method, the data cube has to be divided into blocks. As the experiment result suggests, the block size should not be too small nor too large. Too small block size could achieve higher accuracy but the privacy would be too low. Although large block size could obtain better privacy but the accuracy had to be sacrificed. So the block size has to be carefully chosen.

However, currently there's no practical tool/way which can help to decide the size of block for a data cube to obtain both satisfactory privacy and accuracy.

In this chapter, we will introduce another random data value distortion based method, cubic-wise balance method. This method performs better than zero-sum method in terms of privacy and accuracy without dividing the data cube into blocks.

3.2 Cubic-wise balance method

3.2.1 Terminologies

Definition 5 *In a d-dimensional data cube, a* **unit cube** *is defined as a d-dimensional subcube and the size of each dimension is two. In other words, there are two distinct values in each dimension.*

Definition 6 *A cell* $\Omega[x_1, x_2, ..., x_d]$ *in a unit cube is the* **anchor cell** *if all other cells* $\Omega[y_1, y_2, ..., y_d]$ *in this unit cube satisfy* $x_i \leq y_i \leq x_i + 1$, *where* $1 \leq i \leq d$.

Example 6 *Figure 3.1 gives an illustration of the concepts of data cube, query range, and unit cube. We observe that:*

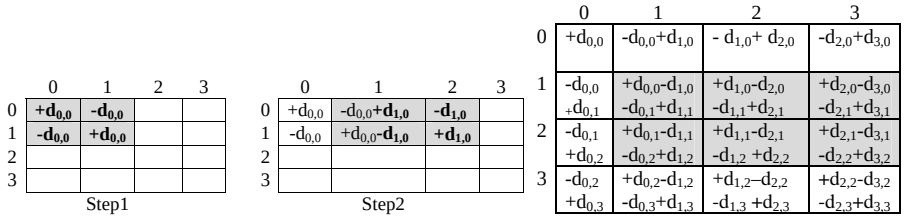
- *This is a 2-dimensional data cube and the size of each dimension is 6.*
- *Cells:* Ω*[3, 1] =102,* Ω*[5, 3] = 41.*
- *The cube sparsity =* $\frac{22}{6\times6}$ *= 61%.*
- *The rectangle with thicker lines indicates the query result of the range query Q(1:4, 1:5).*
- *The three shaded areas are unit cubes. For example, {*Ω*[3, 3],* Ω*[3, 4],* Ω*[4, 3],* Ω*[4, 4]} is a unit cube and* Ω*[3, 3] is the anchor cell of this unit cube.*

3.2.2 2-dimension data cubes

The key idea of the cubic-wise balance method is illustrated as follows. In a unit cube, for each perturbation value, we have a counter-value to cancel its effect, so that

X2 \ X1	0	1	2	3	4	5
0			147			
1		123		102		
2					9	
3	193	5	117	32		41
4	195	103	29		38	
5			49			

Figure 3.1: An illustration of the concepts of data cube, query range, and unit cube.



(a) The First Two Steps.

	0	1	2	3
0	$+d_{0,0}$	$-d_{0,0}+d_{1,0}$	$-d_{1,0}+d_{2,0}$	$-d_{2,0}+d_{3,0}$
1	$-d_{0,0}$ $+d_{0,1}$	$+d_{0,0}-d_{1,0}$ $-d_{0,1}+d_{1,1}$	$+d_{1,0}-d_{2,0}$ $-d_{1,1}+d_{2,1}$	$+d_{2,0}-d_{3,0}$ $-d_{2,1}+d_{3,1}$
2	$-d_{0,1}$ $+d_{0,2}$	$+d_{0,1}-d_{1,1}$ $-d_{0,2}+d_{1,2}$	$+d_{1,1}-d_{2,1}$ $-d_{1,2}+d_{2,2}$	$+d_{2,1}-d_{3,1}$ $-d_{2,2}+d_{3,2}$
3	$-d_{0,2}$ $+d_{0,3}$	$+d_{0,2}-d_{1,2}$ $-d_{0,3}+d_{1,3}$	$+d_{1,2}-d_{2,2}$ $-d_{1,3}+d_{2,3}$	$+d_{2,2}-d_{3,2}$ $-d_{2,3}+d_{3,3}$

(b) Perturbation Results.

Figure 3.2: An Illustration of the cubic-wise balance method on 2-dimensional data cube.

the summation of the distortions within the unit cube is maintained at zero. In this section, we present how the cubic-wise balance method works on 2-dimensional data cubes.

The working mechanism of the cubic-wise balance method for perturbing 2-dimensional data with size $m \times n$ is described as the following.

- Suppose $\Omega[x_1, x_2]$ is a non-null cell of a 2-dimensional data cube, where $0 \leq x_1 < m$, $0 \leq x_2 < n$, and $\Omega[x_1, x_2] = t$. Let $\Omega[x_1, x_2]$ be an anchor cell which decides a unit cube, then other cells in the same unit cube are $\Omega[y_1, y_2]$, where $x_1 \leq y_1 \leq x_1 + 1$, and $x_2 \leq y_2 \leq x_2 + 1$. Let the relative perturbation range be $[0, \Delta]$, where $\Delta > 0$. For each unit cube, this method generates a random data α where $-|t|\Delta \leq \alpha \leq |t|\Delta$ and assigns $+\alpha$ to the anchor cell $\Omega[x_1, x_2]$. The method assigns $+\alpha$ to $\Omega[y_1, y_2]$ if $((y_1 + y_2) - (x_1 + x_2)) \bmod 2 = 0$, otherwise assigns $-\alpha$ to $\Omega[y_1, y_2]$.

Example 7 *Figure 3.2 shows a* 4×4 *2-dimensional data cube. Assume that all cells are non-null cells and the relative perturbation range is* $[0, \Delta]$ *where* $\Delta > 0$.

Let us start from cell $\Omega[0,0]$*, which decides a unit cube containing four cells:* $\Omega[0,0]$*,* $\Omega[0,1]$*,* $\Omega[1,0]$*, and* $\Omega[1,1]$*. If* $d_{0,0}$ *is a random value generated within the range* $[-|\Omega[0,0]|\Delta, |\Omega[0,0]|\Delta]$*, the perturbation added to this unit cube is shown on the left side of Figure 3.2(a). In the figure, the gray area indicates the unit cube. Next, we consider cell* $\Omega[1,0]$*, which decides a unit cube shown as the gray area on the right side of Figure 3.2(a), where* $d_{1,0}$ *is a random value generated within the range* $[-|\Omega[1,0]|\Delta, |\Omega[1,0]|\Delta]$.

Once we have the same perturbation process for all the cells, the perturbed data set is shown in Figure 3.2(b). As illustrated, unlike the random value perturbation, the perturbation approach in the cubic-wise balance method is not random but a constraint perturbation. As shown in Figure 3.2(b), after perturbation, each cell is added several perturbations and the final perturbation of a cell is the sum of all these perturbations. Therefore, different cells end up with very different perturbation values.

Now the question is how the cubic-wise balance method guarantees high accuracies for range-sum queries? To illustrate this, let us consider a range-sum query $Q(1:3, 1:2)$ *on the perturbed data shown in Figure 3.2(b). In the figure, the query result is indicated by the gray area. Assume that* $S(Q)$ *is the true answer for the query* Q *based on actual data and* $S'(Q)$ *is the answer based on perturbed data, then* $S'(Q) - S(Q) = d_{0,0} - d_{3,0} - d_{0,2} + d_{3,2}$*, since most of the perturbations on the cells belonging to this query are cancelled with each other. As a result, the query result on the perturbed data is very close to the query result on the actual data, especially for large queries.*

3.2.3 Multi-dimensional data cubes

The cubic-wise balance method can be easily extended to deal with d-dimensional data. In this case, the unit cube is also d-dimension and the size of each dimension is two. In order to cancel the effect of perturbation added to one cell, the perturbation values assigned to any two neighboring cells have opposite signs. More specifically, if the perturbation $+\alpha$ is assigned to $\Omega[x_1, x_2, ..., x_d]$ which is the anchor cell, then the perturbation $+\alpha$ is assigned to $\Omega[y_1, y_2, ..., y_d]$ if $(\sum y_i - \sum x_i)$ is an even number; otherwise, $-\alpha$ is assigned to $\Omega[y_1, y_2, ..., y_d]$.

Figure 3.3 shows the pseudocode of the perturbation algorithm of the cubic-wise balance method. Essentially, this algorithm is an iterative process. For each iteration, all cells in a unit cube are perturbed. Non-anchor cells are handled separately as shown in Figure 3.4.

```
Input:     A d-dimensional data cube.
           A relative perturbation range, Δ.
Output:    A perturbed data cube.
Parameter: suffix[0..d-1]: the current location of a cell
           dim_size[0..d-1]: the size of each dimension
           curr_dim: the current dimension
Procedure Perturb(suffix, curr_dim)
 1.    If (curr_dim==d-1)
 2.         ω ← the value of the anchor cell identified from suffix;
 3.         If (ω is null) return;
 4.         α ← get_perturbation(ω, Δ); //generate perturbation data value.
 5.         //Non_anchor_cells() deal with non-anchor cells. Details refer to Figure 3.4
 6.         Non_anchor_cells(suffix, suffix1, α);
 7.         return;
       End If
 8.    For ( i = 0; i<dim_size[curr_dim]-1; i++)
 9.         suffix[curr_dim] ← i;
10.         Perturb(suffix, curr_dim+1);
       End for
```

Figure 3.3: The perturbation algorithm of the cubic-wise balance Method

```
Input:     An unit cube
Output:    A perturbed unit cube
Parameter: anchor_suffix[0..d-1]: the suffix of the anchor cell
           suffix1[0..d-1]: the suffix of non-anchor cell
           curr_dim: the current dimension
           perturbation: return value of line 4 in Figure 3.3
Procedure: Non_anchor_cells(anchor_suffix,suffix1,curr_dim, perturbation)
 1.    If(curr_dim==d-1)
 2.        For(i =0; i < d-1; i++) { sum1 += anchor_suffix[i]; sum2 += suffix1[i];}
 3.        If(mod(sum2-sum1)!=0) perturbation = -perturbation;
 4.        add perturbation to corresponding cell;
 5.        return;
       End If
 6.    For(j = anchor_suffix[curr_dim]; j≤anchor_suffix[curr_dim]+1; j++)
 7.        suffix1[curr_dim] = j;
 8.        curr_dim++;
 9.        Non_anchor_cells(anchor_suffix,suffix1,curr_dim);
       End For
```

Figure 3.4: The algorithm for data perturbation in non-anchor cells of a unit cube

3.2.4 Error bound analysis

Error bound refers to the maximum possible difference between the actual result of a range-sum query Q and the result of the same range-sum query on the perturbed data cube. In this subsection, we present the error bound of zero-sum method is also valid for the cubic-wise balance method. In additional, we give the maximum value of the error which might be created by cubic-wise balance method.

Given a 2-dimensional dataset, suppose all perturbations $d_{i,j}$ are random data uniformly distributed within $[-\alpha, \alpha]$ where $\alpha > 0$. Given a range query $Q(l_1 : h_1, l_2 : h_2)$, S(Q) denotes the summation of query Q before perturbation while S'(Q) denotes the summation of the same query.

As illustrated in Figure 3.2(b), we have known that the final perturbation $D_{i,j}$ of cell[i,j] is the summation of several perturbations:

$$D_{i,j} = d_{i-1,j-1} - d_{i-1,j} - d_{i,j-1} + d_{i,j}$$

Then

$$\begin{aligned}
S'(Q) - S(Q) &= \sum_{i=l_1}^{h_1}\sum_{j=l_2}^{h_2} D_{i,j} = \sum_{i=l_1}^{h_1}\sum_{j=l_2}^{h_2}(d_{i-1,j-1} - d_{i,j-1} - d_{i-1,j} + d_{i,j}) \\
&= \sum_{i=l_1}^{h_1}\{(d_{i-1,l_2-1} - d_{i,l_2-1} - d_{i-1,l_2} + d_{i,l_2}) + (d_{i-1,l_2} - d_{i,l_2} - d_{i-1,l_2+1} + d_{i,l_2+1}) + \ldots \\
&\quad + (d_{i-1,h_2-1} - d_{i,h_2-1} - d_{i-1,h_2} + d_{i,h_2})\} \\
&= \sum_{i=l_1}^{h_1}(d_{i-1,l_2-1} - d_{i,l_2-1} - d_{i-1,h_2} + d_{i,h_2}) \\
&= (d_{l_1-1,l_2-1} - d_{l_1,l_2-1} - d_{l_1-1,h_2} + d_{l_1,h_2}) + (d_{l_1,l_2-1} - d_{l_1+1,l_2-1} - d_{l_1,h_2} + d_{l_1+1,h_2}) + \ldots \\
&\quad + (d_{h_1-1,l_2-1} - d_{h_1,l_2-1} - d_{h_1-1,h_2} + d_{h_1,h_2}) \\
&= d_{l_1-1,l_2-1} - d_{l_1-1,h_2} - d_{h_1,l_2-1} + d_{h_1,h_2}
\end{aligned} \tag{3.1}$$

Since $d_{i,j}$ are mutually independent random variables, $E[d_{i,j}] = 0$ and $|d_{i,j}| < \alpha$. According to Theorem 2 in Chapter 2, for $a > 0$,

$$Pr[(S'(Q) - S(Q)) > a] < e^{-a^2/2\sum \alpha^2} = e^{-a^2/8\alpha^2}$$

Furthermore, from equation 3.1, we can derive the maximum error which could be created by perturbed data for cubic-wise balance method.

Theorem 3 *Given a 2-dimensional dataset, suppose all perturbations $d_{i,j}$ are random data uniformly distributed within $[-\alpha, \alpha]$ where $\alpha > 0$. Given a range query Q,*

S(Q) denotes the summation of query Q before perturbation while S'(Q) denotes the summation of the same query, then $|S(Q) - S'(Q)| \leq 4\alpha$

Proof: As illustrated in equation 3.1,

$$S'(Q) - S(Q) = d_{l_1-1,l_2-1} - d_{l_1-1,h_2} - d_{h_1,l_2-1} + d_{h_1,h_2}$$

Since $-\alpha \leq d_{i,j} \leq \alpha$, we get

$$|d_{l_1-1,l_2-1} - d_{l_1-1,h_2} - d_{h_1,l_2-1} + d_{h_1,h_2}| \leq 4\alpha$$

Thus $|S(Q) - S'(Q)| \leq 4\alpha$ $\square$

Theorem 3 indicates that, for any range-sum query over 2-dimensional data cubes, the upper error bound is a constant value, which is solely determined by the upper bound of the perturbation range.

For 3-dimensional data cubes, we can also derive the final perturbation value of cell $[i, j, k]$ as:

$$D_{i,j,k} = -d_{i-1,j-1,k-1} + d_{i-1,j-1,k} + d_{i-1,j,k-1} +$$
$$d_{d,j-1,k-1} - d_{i,j,k-1} - d_{i,j-1,k-1} - d_{i-1,j,k} + d_{i,j,k}$$

Similarly, we can derive $|S(Q) - S'(Q)| \leq 2^3\alpha$. In addition, for a perturbed d-dimension dataset, the upper error bound of a range-sum query is given by the following corollary.

Corollary 2 *For a d-dimensional dataset, under the assumption of Theorem 3,* $|S(Q) - S'(Q)| \leq 2^d\alpha$

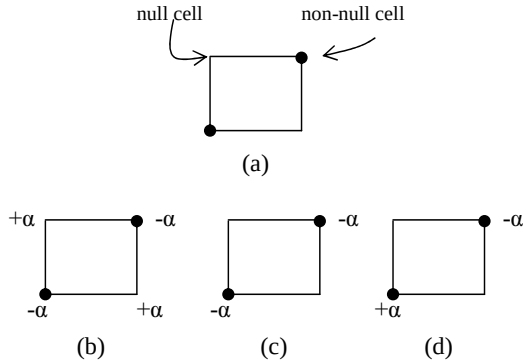


Figure 3.5: 2-dimensional unit cube with 2 null cells

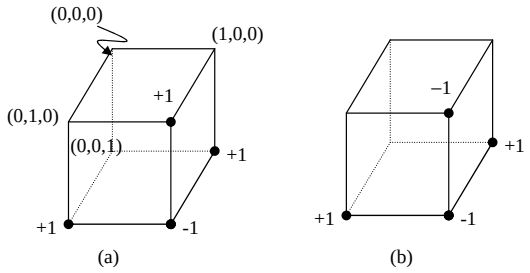


Figure 3.6: 3-dimensional unit cube with 4 null cells

3.3 Handling null cells

Before we present the method of how cubic-wise balance method handle null cells, let's see an example first.

Example 8 *Given a 2-dimensional unit cube, the anchor cell is at the up-left corner. In the unit cube, there are two null cells and two non-null cells, as shown in Figure 3.5(a). Suppose the perturbation value for this unit cube is* α.

Figure 3.5(b) is the perturbation result without considering the presence of null cells. As shown, two null cells become non-null cells with value $+\alpha$.

One simple method to deal with null cells is to let null cells still be null after the perturbation process. The result is shown in Figure 3.5(c). The perturbations added to the unit cube cannot cancel each other. To deal with this problem, we change one

$-\alpha$ to $+\alpha$. The result is shown in Figure 3.5(d), the sum of perturbations added to this unit cube is still zero.

Since the perturbation values added to a specific unit cube are the same, only the signs (+ or -) of values are different. The problem of how to efficiently counterbalance the perturbations added to a unit cube is actually equivalent to the problem of how to assign the sign of perturbations.

3.3.1 Sign assignment problem

The problem on *how to optimally assign the sign of perturbations to a unit cube* can be described as follows.

Sign Assignment Problem (SAP): *Given a subset S of a d-dimensional unit cube, the SAP is to find an assignment of signs to S, such that the absolute value of the sum of signs for all range queries is minimal.*

Let Q be a range query in a unit cube. The absolute value of sum of signs of all cells in the query Q is equal to $H_s(Q)$:

$$H_s(Q) = |\sum_{\omega_i \in Q} sign(\omega_i)| \tag{3.2}$$

Where $sign(\omega_i) = +1$ if ω_i is assigned a positive sign, otherwise $sign(\omega_i) = -1$. Therefore, the problem can be formulated as the following optimization.

$$min_{\{sign(\omega_i)|\omega_i \in S\}}(\sum_{allQ} H_s(Q))$$

If an assignment for S satisfies $H_s(Q) = 1$ if the size of Q is odd and $H_s(Q) = 0$ if the size of Q is even, this assignment is a strict optimal problem. However, the strict optimal solution may not always be achievable.

Example 9 *Let S = {(1,1,0), (1,0,1), (0,1,1), (1,1,1)}. Figure 3.6(a) is the result of simply forcing the original null cells to be null after the perturbation process. Figure 3.6(b) shows a better solution to S. In Figure 3.6(a), there are three + and only one -, thus the sum of perturbations in the unit cube is $+2\alpha$ (suppose α is the perturbation value). However, in Figure 3.6(b), the number of + and - is the same, so the sum of perturbations in the unit cube is zero.*

Although Figure 3.6(b) is an optimal solution, it is not a strict optimal solution, since there exists a range query Q(1, 1, 0:1) such that $H_s(Q) = 2$.

3.3.2 NP-completeness of the SAP

Here, we illustrate the NP-completeness of the SAP. By suitable transformation, SAP is equivalent to PPS (Partial Parallel Searchability) problem which was proposed by Srinivasan in [40].

We firstly prove that SAP is in NP, and then reduce PPS problem to SAP.

3.3.2.1 SAP is in NP For 2-dimensional data cube, a unit cube has 4-cells, the number of assignment of signs is at most 16. For d-dimensional data cube, a unit cube has 2d cells, and each cell could be assigned to a positive sign or a negative sign, so the number of all possible sign assignment is 2^{2^d}.

If considering null cells, for a d-dimensional unit cube, suppose it has p null cells, so the number of non-null cells is $2^d - p$, the number of sign assignment is 2^{2^d-p}.

So $SAP \in NP$.

3.3.2.2 Reduce PPS to SAP

PPS problem:Let the database consists of segment type $S = \{S_1, S_2, ..., S_n\}$ and the query categories that can occur be $Q = \{q_1, q_2, ..., q_m\}$. PPS is to find an optimal distribution of S into k nodes, such that the number of searches to answer all queries once is minimum.

Suppose $S_p = \{(S_{p1}), (S_{p2}), ..., (S_{pk})\} where((S_{pi}) \in S)$ be one of the possible partition of S into k parts. The number of searches required to retrieve the segments required for the jth query q_j in the above partition is

$$max_{1 \leq i \leq k} |(S)_j \cap S_{pi}|$$

where $(S)_j$ denotes the set of segments required by the jth query. Therefore, the total number of searches required to answer all queries once is

$$\sum_{j=1}^{m} max_{1 \leq i \leq k} |(S)_j \cap S_{pi}|$$

WPPS problem:If instead of considering number of searches, we concern the minimal cost to answer all queries once, PPS can be transformed to WPPS (weighted Partial Parallel Searchability Problem).

Let the database consists of segment type $S = S_1, S_2, ..., S_n$ and the query categories that can occur be $Q = q_1, q_2, K, q_m$.

Suppose $S_p = \{(S_{p1}), (S_{p2}), ..., (S_{pk})\} where((S_{pi}) \in S)$ be one of the possible partition of S into k nodes. Function f_w is used to calculate the weight of each node, $W_{pi} = f_w(S_{pi})$. So $W_p = (W_{p1}), (W_{p2}), ..., (W_{pk})$ is the weights for assignment S_p . WPPS is to find an optimal distribution of S into k nodes, such that the total weight (cost) to answer all queries once is minimum.

The total weights required to retrieve the segments required for the jth query q_j in the above partition is

$$min|\sum_{1\leq i\leq k}(W)_j \cap W_{pi}|$$

Where $(W)_j \cap W_{pi} = W_{pi}$ if query q_j access pith node, otherwise, $(W)_j \cap W_{pi} = 0$. Therefore, the minimal weight of answer all queries is

$$\sum_{j=1}^{m} min|\sum_{1\leq i\leq k}(W)_j \cap W_{pi}|$$

If we reduce WPPS by limit $W_{pi} \in -1, 0, 1$, it is equivalent to SAP. So PPS can be reduced to SAP.

3.3.3 A heuristic assignment algorithm for SAP

Since the SAP is an NP-complete problem, we propose a Heuristic Assignment Algorithm (HAA) to deal with this problem.

The HAA is based on the observation of the 2-dimensional plane:

1. if and only if there are only two non-null cells and the two cells are at the ends of a diagonal line (as shown in Figure 3.5 (a)), the two cells should be assigned opposite signs;

2. For all cells do not satisfied with above condition, two neighboring cells should be assigned opposite signs in order to minimize the sign sum of a query.

So the HAA first deals with all 2-dimensional planes of a range query, then process the rest unassigned non-null cells whose signs will be decided by their neighboring cells. The pseudocode of HAA is presented in Figure 3.7.

The computation cost of the HAA algorithm is mainly on examining all 2-dimensional planes of a d-dimensional unit cube. Thus the complexity of HAA algorithm is

Input: A unit cube Ω
Output: Sign assignment for the unit cube
Procedure: HAA(Ω)

```
1.  For each cell ω_i, let sign(ω_i) = 0
2.  For each 2-dimensional plane of a unit cube
    {
3.      If (there are only two non-null cells ω_i, ω_j) and ( |ω_i ⊕ ω_j| = 2)
        /*the two cells lie at the ends of a diagonal line*/
4.          If (the two cells haven't been assigned a sign)
5.          Then {assign + and - to two cells respectively}
6.          Else if (one cell has been assigned a sign)
7.              Then {assign an opposite sign to the other cell}
    }
8.  For each of the rest unassigned non-null cell ω_i
9.          If (the cell is the last one)
10.         Then {If (∑ sign > 0) //(∑ sign is the sum of all signs
11.             Then {sign(ω_i) = -1}
12.             Else {sign(ω_i) = +1}}
13.         Else {
14.             sum_s = ∑ sign(ω_j) where |ω_i ⊕ ω_j| = 1
15.             If (sum_s > 0) Then {sign(ω_i) = -1}
16.             Else {sign(ω_i) = +1}
            }
    End For
```

Figure 3.7: The heuristic assignment algorithm(HAA) for SAP

bounded by $2^{d-2}C_d^2$, which is the number of 2-dimensional planes. Since it is unnecessary to apply HAA algorithm to null unit cubes (all cells of the unit cube are null) , the sparser the data cube is, the smaller the overhead of the HAA algorithm will have.

To measure the performance of the HAA algorithm, we provide a baseline algorithm, called the Basic Assignment Algorithm (BAA), which simply forces null cells to be null after the perturbation process (as shown in Figure 3.5 (c)). BAA is simple, but the two opposite signs assigned to non-null cells cannot be counterbalanced with each other and potentially will degrade the query accuracy.

To evaluate the performance of sign assignment algorithms, we employed a measure: $\overline{H_s}(Q^d)$:

$$\overline{H_s}(Q^d) = \frac{H_s(Q^d)}{number\ of\ queries} \tag{3.3}$$

where

$$H_s(Q^d) = \sum_{all\ Q^d} | \sum_{\omega_i \in Q^d} sign(\omega_i)| \tag{3.4}$$

For a given unit cube, $H_s(Q^d)$ calculates the sum of all signs of all d-dimension range queries (denoted as Q^d). $\overline{H_s}(Q^d)$ indicates the average sign sum of all d-dimension queries. Therefore, the smaller $\overline{H_s}(Q^d)$ is, the better the solution is.

In addition, we compare $H_s(Q)$(defined in Formula 3.2) of HAA and BAA for each query and see how many queries favor BAA or HAA, respectively. For a specific query, if $H_s(Q)_{BAA} > H_s(Q)_{HAA}$, HAA is better; if $H_s(Q)_{BAA} < H_s(Q)_{HAA}$, BAA is better; otherwise they perform equally. For example, Figure 3.5(a) is a subset of a 2-dimensional unit cube, S = $\{(0,1),(1,0)\}$. With BAA, the sign assignment for S is shown in Figure 3.5(c), and with HAA, the sign assignment for S is shown in Figure 3.5(d). Given a range query Q = $(0:1, 0:1)$, the absolute value of sum of signs for BAA is 2 while the absolute value of the sum of signs for HAA is 0. Thus, for the query Q, HAA is considered better than BAA.

Case Study. For a 3-dimensional unit cube, there are 2^3 cells, each cell could be null or non-null. In total, there are $2^{2^3} = 256$ possible cases. In other words, a 3-dimensional unit cube has 256 distinct subsets.

Furthermore, for a given 3-dimensional unit cube, $|Q^1| = 12$ (there are twelve 1-dimension queries), $|Q^2| = 6$ and $|Q^3| = 1$.

We calculate $H_s(Q^d)$ for all possible d-dimension range queries over all possible 3-dimensional unit cubes.

Table 3.1 shows the performance of BAA and HAA for 3-dimensional unit cubes. It showed that HAA performs better than BAA except for 1-dimension query. For all 3-dimension queries, the average value of the sign sum of a query was only 0.5 when using HAA, but the value increased to larger than 1 when using BAA.

Table 3.2 shows the performance of each algorithm on queries. Although BAA was slightly better for 1-dimension queries, HAA performed much better for higher dimension queries. 11.72% of 2-dimension queries favored HAA while only 4.69% of 2-dimension queries favored BAA.

Table 3.1: $\overline{H_s}$ of 3-dimensional unit cube

$\overline{H_s}$	Q^1	Q^2	Q^3
BAA	0.5	0.75	1.0938
HAA	0.5736	0.6094	0.5

Table 3.2: Queries of 3-dimensional unit cube

	Q^1	Q^2	Q^3
BAA *is better*	3.68%	4.69%	0
HAA *is better*	0	11.72%	28.91%

Table 3.3 and Table 3.4 present the performance results of 4-dimensional unit cube. As the number of query dimension increased, the performance of HAA improved significantly. For 4-dimensional query, if using BAA, the average sign sum of a query was 1.57. This was reduces to 0.57 if using HAA. Also over 40% queries favored HAA while only 0.77% queries favored BAA.

Table 3.3: $\overline{H_s}$ of 4-dimensional unit cube

$\overline{H_s}$	Q^1	Q^2	Q^3	Q^4
BAA	0.5	0.75	1.0938	1.571
HAA	0.5923	0.6587	0.723	0.5681

Table 3.4: Queries of 4-dimensional unit cube

	Q^1	Q^2	Q^3	Q^4
BAA is better	4.61%	6.19%	5.84%	0.77%
HAA is better	0	10.76%	24.32%	43.96%

3.4 Experiment results

The experiments were done on the same data set mentioned in Section 2.6. The performance measurements are privacy factor and accuracy factor.

3.4.1 The effect of distortion range

First, we illustrate the effect of different distortion ranges on privacy preservation. In this experiment, we fixed the data cube sparsity at 60% and fixed the upper bound of distortion range at 50% while increasing the lower bound of distortion range from 0% to 40%. Then we observed the achieved average privacy.

Figure 3.8 shows the privacy performance of the cubic-wise balance method, zero-sum method and the random data distortion method when the lower bound of distortion range is increased. In the figure, it was not surprising to see a trend that the privacy values of both methods increased with the increase of the upper bound of the relative distortion ranges, since higher distortion can bring better privacy preservation. Also, we observed that the achieved privacy of the cubic-wise balance method was consistently better than that of the zero-sum method and random distortion method. We also observed that the change of distortion ranges affected the privacy of the cubic-wise balance method much more than the other two methods. When the distortion range increases from [0, 50%] to [40%, 50%], the privacy of the random distortion method increased from 0.24 to 0.43, the privacy of the zero-sum method increased from 0.5 to 0.79, while the privacy of the cubic-wise balance method in-

creased from 2.7 to 4.5. Finally, with the same perturbation range, the privacy of the cubic-wise balance method was much higher than that of the traditional perturbation method, especially when the perturbation range was large.

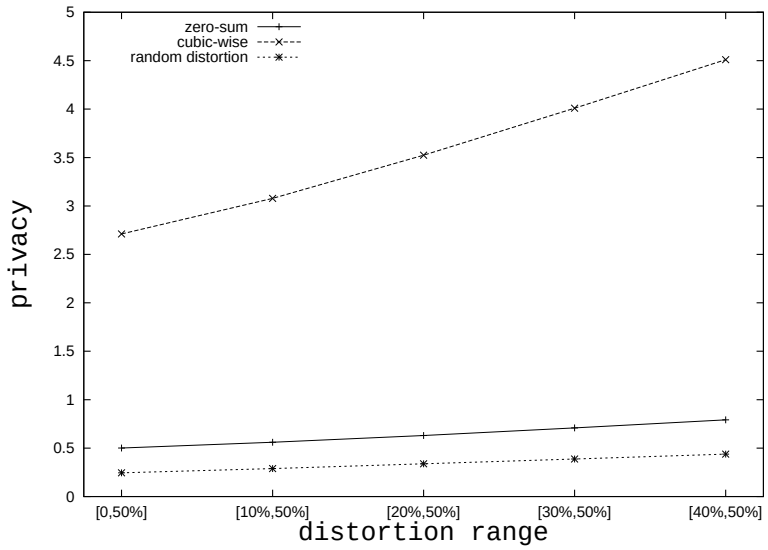


Figure 3.8: The effect of different perturbation ranges on privacy preservation.

3.4.2 The effect of privacy on query accuracy

Since higher privacy means larger deviation of the distorted value from the original value, distorted data cube with higher privacy would result in lower accuracy for the same range-sum query. So this experiment compared the query accuracy of the three methods at the same level of privacy. Because it is almost impossible to get exactly the same privacy value for different methods, we treat all privacy values falling into the same small range as the same level. For example, if the privacy values of all different methods fall into the small range [0.8, 0.9], we considered these values were at the same level. In this experiment, the data cube sparsity was fixed at 60%. Also, we generated 600 range-sum queries with the query size between 50 and 1000. The average query accuracy of these queries was reported and compared.

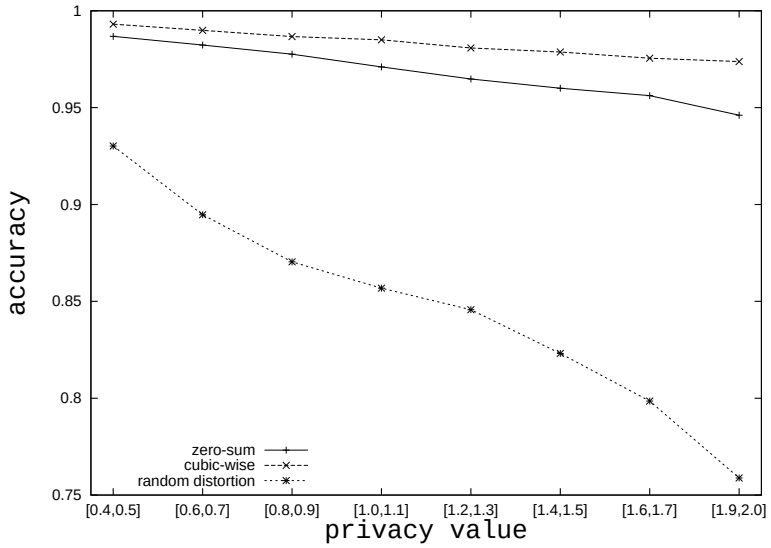


Figure 3.9: The effect of different privacy values on query accuracy.

Figure 3.9 shows the accuracy of the cubic-wise balance method, zero-sum method and random distortion method with the change of privacy values. As can be seen, the accuracy of cubic-wise balance method and zero-sum method were significantly and consistently better than that of the random distortion method. And the accuracy of cubic-wise balance method was slightly better than that of zero-sum method.

Also, with the increase of privacy values, the accuracy of the random distortion method decreased dramatically. In contrast, the accuracy of the cubic-wise balance method and zero-sum method were not much affected by the change of privacy values.

3.4.3 The effect of data cube sparsity

To test the effect of data cube sparsity on the performance of the cubic-wise balance method, we fixed the perturbation range as [10%, 15%] for cubic-wise balance method, [20%, 70%] for zero-sum method and [0%, 140%] for random distortion method. In this experiment, we generated 600 range-sum queries with the query size between 50 and 1000.

Figure 3.10(a) and Figure 3.10(b) show the achieved privacy values and accuracy values of the three methods with the change of data cube sparsity. As can be seen, The privacy of random distortion method was almost not affected by the change of cube sparsity and the achieved privacy of cubic-wise balance method and zero-sum method were consistently better than the privacy of random distortion method when the sparsity was not larger than 70%. Furthermore, the privacy of cubic-wise balance method was larger and increased faster than those of zero-sum method.

Figure 3.10(b) shows that the accuracy of the cubic-wise balance method and zero-sum method were significantly and consistently better than that of the random distortion method at different data cube sparsity. Cubic-wise balance method aslo performs better than zero-sum method. The accuracy value of zero-sum method fell in the range [95.6%, 96%] while the accuracy value of cubic-wise balance method fell in the range [97.3%, 98.8%]

Another observation was that, the lower sparsity the data cube was, the better that the accuracy can be achieved for each method evaluated.

3.4.4 The effect of query size

Here, we illustrate the effect of changing query sizes. In this experiment, we fixed the data cube sparsity at 60% and kept privacy values at the same level, [0.4, 0.5]. The query size is changing from small (< 50) to large($[500, 1000]$). We generated 600 range-sum queries for each range of query size and compared the average query accuracy of these queries.

Figure 3.11 shows the accuracy performance of the three methods when the query size is increased. As can be seen, the accuracy of the cubic-wise balance method and zero-sum method were significantly and consistently better than that of the random

distortion method. The accuracy of the random distortion method depended a lot on the query size. When the query size was small, the accuracy was as low as 87%. Only when answering large range queries, the traditional perturbation method could obtain relatively high accuracy. In contrast, the accuracy of zero-sum method was always higher than 94%, and the accuracy of cubic-wise balance method was consistently over 98.5% even for small query size.

In addition, Figure 3.11 shows that the cubic-wise balance method performs consistently better than zero-sum method in estimating the answer of range-sum queries. Furthermore, the size of query affected the performance of zero-sum method more than that of cubic-wise balance method. When the size of queries were $[500, 1000]$ or larger, the accuracy of both methods were higher than 99.5%. However, when the query size decreased to smaller than 50, the accuracy of zero-sum method was only 94% while the accuracy of cubic-wise balance method was about 98.5%.

3.4.5 A summary of experimental results

Since the experiment results of random distortion method and zero-sum method were consistent with the experiment results shown in Section 2.6, here we only summarize the results of cubic-wise balance method with zero-sum method.

- The distortion range affected cubic-wise balance method more than zero-sum method. With the same distortion range, cubic-wise balance method result in higher privacy value. In addition, with the increase of distortion range, the privacy of cubic-wise balance method increased faster than that of zero-sum method.
- The privacy of cubic-wise balance method was also more sensitive to cube sparsity than that of zero-sum method.

- Under the same condition, the accuracy of cubic-wise balance method was always better than that of zero-sum method. The reason is that with zero-sum method, the inaccuracy is created by partially covered blocks. For a d-dimensional range query of a d-dimensional cube, there could be 2^d partially covered blocks producing the inaccuracy. While with cubic-wise balance method, the only 2^d cells at each corner of the d-dimensional range query would create inaccuracy.

- Cubic-wise balance method performed better for small size queries. This was because the distortions added to cells are counterbalanced by their neighbor cells. While in zero-sum method, counterbalancing the distortions added to cells depends on how many cells in a row are included in a range query. The more, the better.

The high accuracy achieved by the cubic-wise balance method is due to the fact that the way of cubic-wise method adding perturbations to cells enables most of the perturbations of a range-sum query be counter-balanced. In other words, the difference between the answer of a range-sum query based on perturbed data and the answer based on actual data is small, thus the accuracy loss is small.

3.5 Cubic-wise balance method vs. zero-sum method

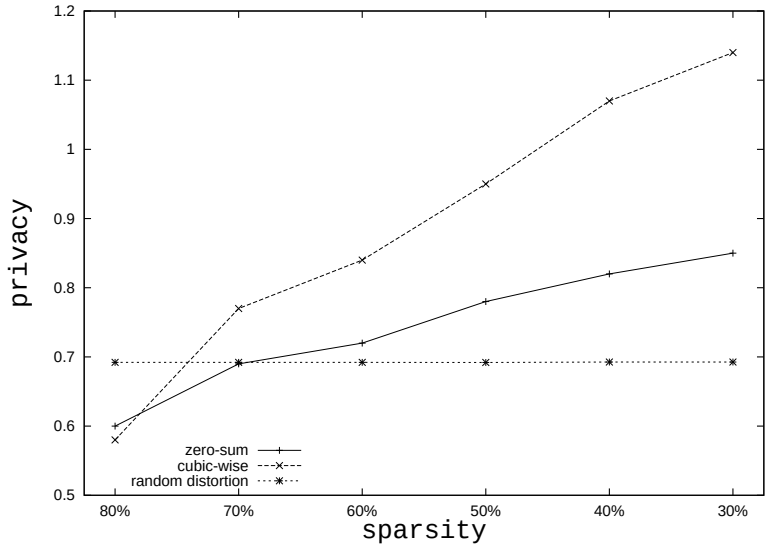
From Section 3.4.5, we know that in general the cubic-wise balance method performed better than zero-sum method in terms of data privacy and query accuracy, especially when query size was very small.

In addition, when using zero-sum method, the block size has to be carefully decided. But it is unclear how could administrator obtain the optimal block size. While

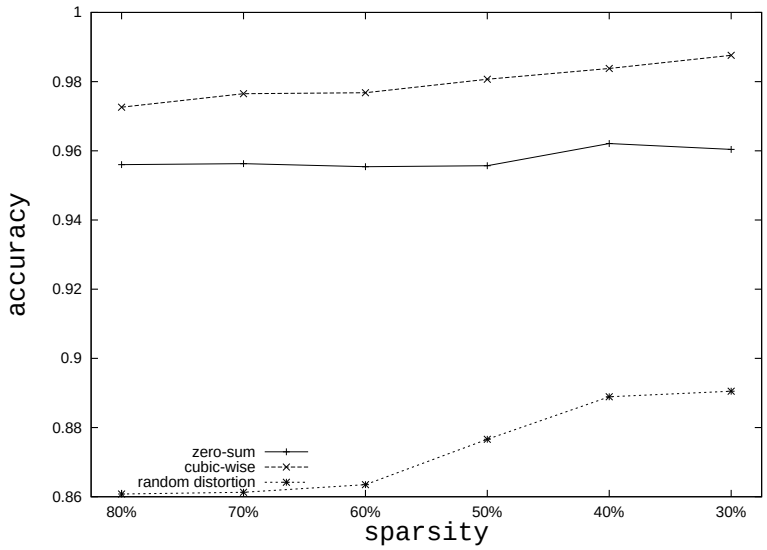
using cubic-wise balance method, it's unnecessary to consider how to partition the data cube into blocks.

On the other hand, zero-sum method is more efficient than cubic-wise balance method. The time complexity of zero-sum method is $d \times n_1 \times n_2 \times ... \times n_d$. while the time complexity of cubic-wise balance method is $2^d \times n_1 \times n_2 \times ... \times n_d$. So the time cost of creating the distorted data cube from an original data cube is smaller if using zero-sum method than using cubic-wise balance method.

Therefore, no one method is overall better than the other.



(a) Privacy preservation



(b) Query accuracy

Figure 3.10: The effect of different cube sparsity

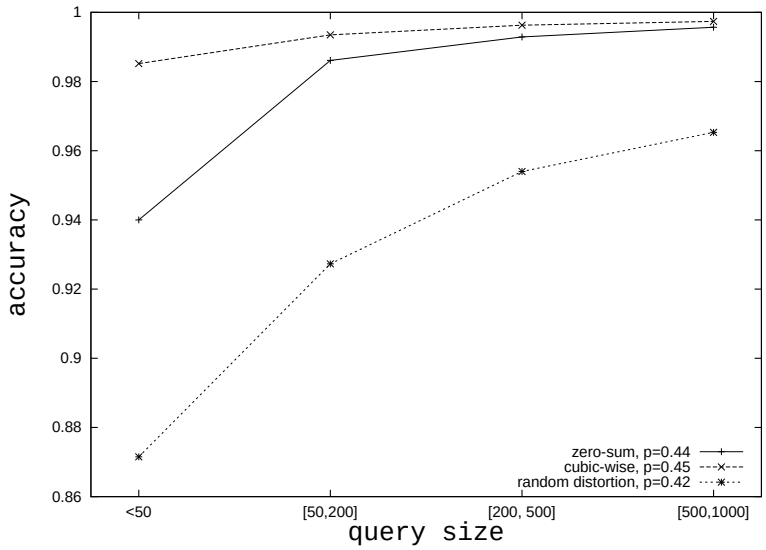


Figure 3.11: The accuracy of different query size

Chapter 4

Extend to Range-max queries

Typically, people think of the range-max query of data cube as the selection of the data cell with the maximum data value. Thus it seems impossible to answer range-max query when original data are deliberately distorted.

However, OLAP query makes heavy use of aggregation for summarizing data since trends derived from the summarized records are more useful for decision-making than individual records themselves [41]. This makes it possible to extend the zero-sum method and cubic-wise method to answer certain range-max queries.

Before we bring up the main idea, we introduce the hierachical structure of data cubes, followed by the data cube operations. Then we present some experiment results and address the limitations of the proposed method.

4.1 An overview on basic concepts

4.1.1 The hierachical structure of data cube

The OLAP concept was proposed for rendering very large, historical (statistical) databases in multidimensional perspectives, and it is oriented to decision making for business users. Many authors have proposed multidimensional data models and query languages [42][43][44]. In literature, multidimensional data are characterized

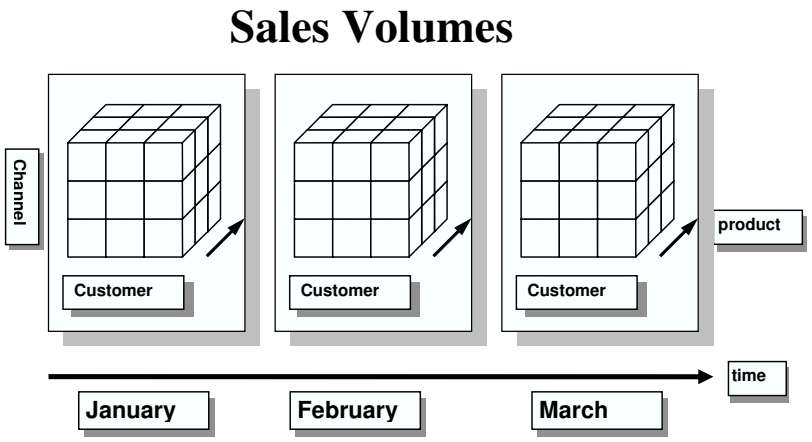


Figure 4.1: 4-dimensional data cube

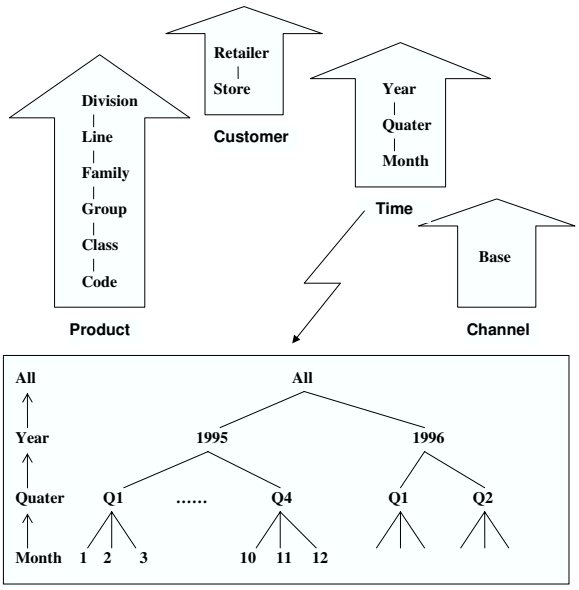


Figure 4.2: Hierarchy structure of data cube

by having two different types of attributes: measure attributes and dimensions (as mentioned in Section 1.1).

Since most proposed models have such constraints as "dimensions are linguistic categories corresponding to different ways of looking at the information", then each dimension is a simple concept hierarchy [45].

A hierarchy is a set of variables which represent different levels of aggregation of the same dimension. A typical example of hierarchy is $Year \rightarrow Quarter \rightarrow Month$.

Example 10 *The data set generated by the APB Benchmark program has four dimensions: customer, product, channel and time. The data model is shown in Figure 4.1. Figure 4.2 shows each dimension in hierarchy.*

The Time dimension is a steep hierarchy containing three levels. Each member of the hierarchy contains at most one parent. Every member of the hierarchy, except the member at the top level, has a parent. Year is the top level and Month is the bottom level.

Similarly, dimension customer has two levels; product has seven levels and channel only has one level. When the number of level larger than one, the upper level can be obtained by aggregating lower level. Thus this data cube has three **aggregation dimensions**.

A hierarchy is an effective form of knowledge representation for encoding prior domain knowledge relevant to data cube. In OLAP environment, hierarchies are used to conceptualize the process of generalizing data as a transformation of values from one domain to values of another domain by means of drill-down/roll-up operator.

4.1.2 Operations of data cube

There are four typical OLAP operations:

1. *Roll-up* – Data is summarized by climbing up the hierarchy or by dimension reduction.

   Example 11 *Consider the hierarchy of Figure 4.2. The roll-up operation allows to change level from Quarter to Year.*

2. *Drill-down* – This is the reverse operation of roll-up. It enables the user to navigate the data cube from a higher-level summary to a lower-level summary or detailed data.

   Example 12 *Consider the hierarchy of Figure 4.2. The drill-down operation allows to pass from Quarter to Month.*

   Through roll-up/drill-down, users can navigate among levels of data ranging from the most summarized to the most detailed.

3. *Slice and dice* – A slice is a subset of a multi-dimensional array which corresponds to a single value for one or more members of the dimensions not in the subset. A dice is a selection of some ranges over the dimensions. The operation of slice and dice helps the user to select one or more dimensions.

   Example 13 *Let us consider the cube of Figure 4.1. Slice the dimension Time deletes this dimension from the cube, thus the result becomes a three-dimension cube.*

   Example 14 *Still consider the cube of Figure 4.1 and suppose that the Year domain is [1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998]. Dice operation allows to cut the part of the domain instances of one dimension of this cube which are specified in the operation. So dice Year = [1990, 1991, 1992] carries out the removal of the above mentioned instances from the domain of the dimension Year, restricting it to the remaining values (1993,...,1998)*

4. *Rotate* – Allows users to change the dimensional orientation of a report or page display. An example of the rotate operation is swapping the rows and columns.

Range-aggregates, i.e., aggregations of the data selected by range conditions are, typical operations in data cube [9]. Rotate will not change the displayed data but only change the view of data. The other operators can be viewed as range queries, and in most cases, the result data set are aggregated data. Picking up the max data value from a set of aggregated data is called *aggregated range-max queries* in this thesis.

4.1.3 Aggregated range-max queries

As defined in Section 2.2, the problem of computing a range-sum query in a d-dimensional data cube can be formulated as follows:

$$sum(l_1 : h_1, ..., l_d : h_d) = \sum_{X_1=l_1}^{h_1} \ldots \sum_{X_d=l_d}^{h_d} \Omega[X_1, X_2, ..., X_d]$$

where l_i and $h_i (1 \leq i \leq d)$ denote the lower and the upper bound of the range query in ith dimension of the data cube.

The range-max query problem can be formulated as getting the *Max_index* of a region of Ω defined as follows:

$$Max\_index(l_1 : h_1, ..., l_d : h_d) = (x_1, x_2, ..., x_d)(1 \leq i \leq d)(l_i \leq x_i \leq h_i)$$

and

$$\Omega[x_1, x_2, ..., x_d] = max\{\Omega[y_1, y_2, ..., y_d] \mid (1 \leq i \leq d)(l_i \leq y_i \leq h_i)\}$$

Problem Definition Let $r_1 = sum(l_{11} : h_{11}, ..., l_{1d} : h_{1d})$,$r_i = sum(l_{i1} : h_{i1}, ..., l_{id} : h_{id})$. The **aggregated range-max query** problem is to find out the $max\{r_i\}$, where r_i is called *sub-range*.

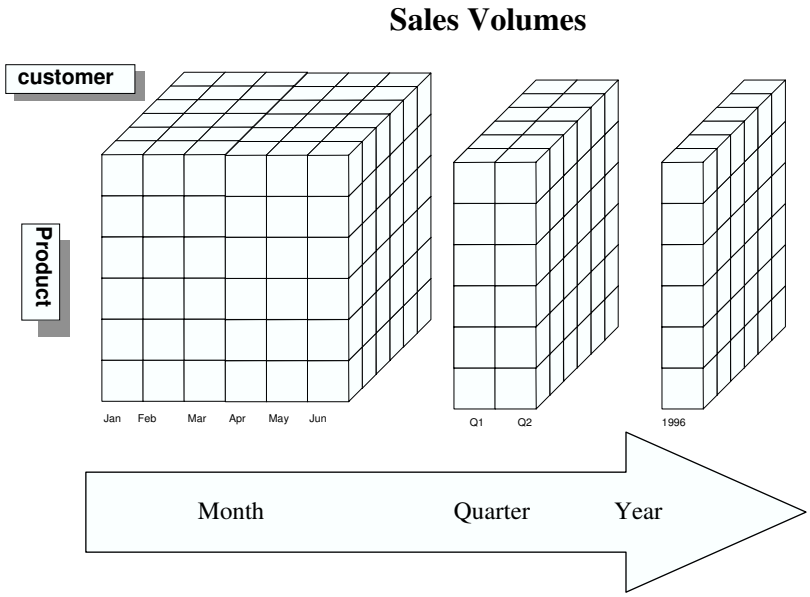


Figure 4.3: Drill-down/roll-up through a dimension

Most of the range-max problem in data cubes can be viewed as aggregated range-max query problem.

Example 15 *Suppose an analyzer tries to do Time Series Analysis. The time is from "January 1995" to "June 1996". Figure 4.3 shows the process of analyzing the sales volumes for a group of time periods over all channels.*

Suppose the right most cube in Figure 4.3 is the result of sales volumes in Year 1996. Next the analyzer tries to figure out which quarter has the maximum sales for a given customer and product in 1996. The drill-down helps to reach the sales data of the two quarters in 1996, and the result is looked like the cube in the middle of Figure 4.3. Each cell in the cube is summarized data corresponding to a range query. For example, the sales data of quarter1 in 1996 is the summation of the sales data of January, February and March in 1996. Thus the maximum data is also a result of range-sum query. The query of selecting the maximum data from a set of aggregated result of range queries is Aggregated range-max Query.

In order to know the sales of each month, the drill-down operations can be further performed on the cube in the middle of Figure 4.3. As long as the drill-down does not reach the bottom level of all dimensions of the data cube, the displayed data value are aggregated values and the range-max query applied to these aggregated data can be viewed as aggregated range-max query.

4.2 Experiment results

4.2.1 Performance measurement

What we concern for the aggregated range-max query is that after deliberately distorting the original data cube, whether the distorted data cube could still preserve the maximum information for range queries.

For example, if in the original data cube, the maximum sales occurs in March 1995 for a given product and a given customer between January 1995 and December 1995. Then in the distorted data cube, when the user issues the same query, the location of the maximum value should be the same as in the original data cube, e.g. the max sales also occurs in March 1995.

Matched query For an aggregated range-max query Q, r_i is a sub-range of Q. Suppose $max\{r_i\}$ is the result of the aggregated range-max query on the original data and $max'\{r_i\}$ is the result of the aggregated range-max query on the distorted data. If $max\{r_i\} = max'\{r_i\}$, we call query Q *matched query* between original data and distorted data.

We use **hit rate** H_r to evaluate the performance of zero-sum method and cubic-wise balance method for aggregated range-max queries.

$$H_r = \frac{the\ number\ of\ matched\ queries}{the\ total\ numberof\ aggregated\ range - max\ queries} \tag{4.1}$$

4.2.2 Method

The experiments were done on the same data set mentioned in Section 2.6.

The APB benchmark proposed three scenarios to evaluate the performance of OLAP applications. The three scenarios are: Budget, Actual and Forecast. The benchmark designed a set of queries for each scenario. Since what we care about is the hit rates of queries, we only do experiments for the Actual scenario. The queries of the other two scenarios are similar to the queries of the Actual scenario except the data sources are different.

The benchmark designed four categories of query for Actual Scenario:

1. Channel sales analysis: this query shows actual dollar sales for a given channel. The product, customer, channel and time period vary with each execution of query.

2. Customer margin analysis: this query shows actual sales for a given customer for the sum of all channels for a requested period. The product, customer and time period vary with each execution of the query.

3. Product inventory analysis: this query shows actual sales for a given product regardless of channel. The product and customer vary with each execution of query.

4. Time series analysis: this query shows actual sales for a given customer and a group of time periods over all channels. The product, customer and time period vary with each execution of the query.

It can be seen that all the four categories of query are aggregated range-max queries if we enforce the aggregation operation MAX on each query.

4.2.3 Results

Thus in this experiment, we distorted the same original dataset by the three distortion methods: random distortion, zero-sum and cubic-wise balance. The privacy of the three distorted datasets were in the range [0.4,0.5]. we generated 600 aggregated range-max queries, then compared the hit rates of the three methods. The size of queries range from < 50 to > 1000. Table 4.1 shows the result of the experiment. The hit rate of cubic-wise balance was the highest while the hit rate of random distortion was the lowest.

Table 4.1: Hit rates of the three methods

	random distortion	zero-sum	cubic-wise balance
hit rate	71.71%	90.67%	93.84%

In order to know whether the query size would affect the hit rate, we partition the query size into five ranges: < 50, $[50, 200]$, $[200, 500]$, $[500, 1000]$ and > 1000. Figure 4.4 shows the result.

We observed that the hit rates of zero-sum method and cubic-balance method were consistently better than that of random distortion method in all size of range queries. However, unlike the accuracy of range-sum queries, it seemed that the hit rate of aggregated range-max queries did not much relate to query size. Because whether a maximum value could be correctly picked up depends on how close the summation of sub ranges were. The closer the summation of sub ranges, the worse hit rates. For example, an aggregated range-max query Q has two sub ranges r_1 and r_2. The summation of these sub ranges on original data set are s_1 and s_2 and the corresponding summation on distorted data set are s'_1 and s'_2. Suppose $max\{s_i\} = s_2$. If s_1 and s_2

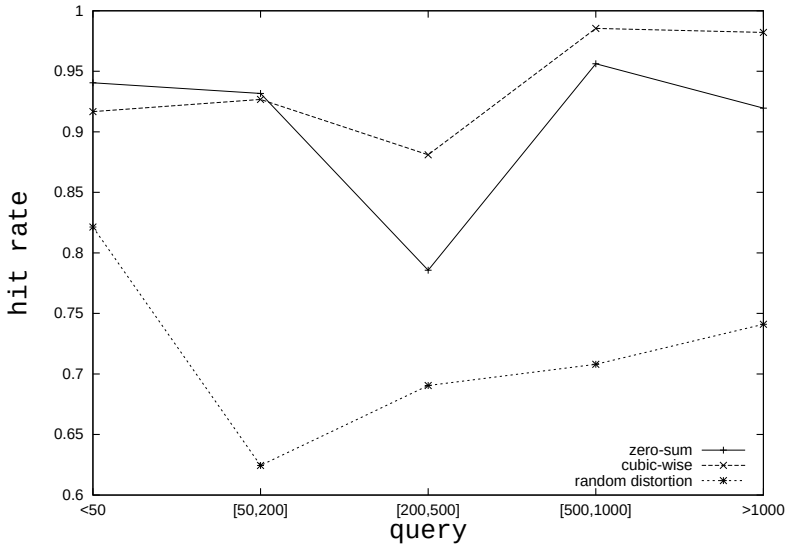


Figure 4.4: The effect of query size on hit rate.

are very closed to each other, it is highly possible that $s'_1 > s'_2$, thus $max\{s'_i\} = s'_1$. The aggregated range-max query Q based on distortion data set could not return the correct answer.

Chapter 5

Conclusion and future work

OLAP enables analyzers and managers to gain insight into the performance of the enterprise and help decision making. Thus the individual data is valuable and sensitive and should be protected from being revealed. With the development of OLAP, the users now include customers, partners or even third parties. This makes the privacy concern become more and more important. Notice the fact that OLAP queries make heavy use of aggregation for summarizing data since trends derived from the summarized records are more useful for decision-making rather than individual records themselves, our objective was to develop a method which can accurately estimate the answer for range-sum queries without release the actual individual data value.

In this thesis, we proposed two random data distortion based method for data cubes. The experiment results showed that the proposed methods, zero-sum method and cubic-wise balance method, can help us achieve the above goal.

The accuracy of zero-sum method and cubic-wise balance methods were consistently and significantly better than that of random data value distortion method. The reason is that both methods try to cancel the effect of distortions added to cells. For zero-sum method, it adjusts the marginal sum of distortions in each block to zero. Thus the effects of distortions added in those blocks which are fully covered by a range query are counterbalanced, and only those partially covered blocks may create inaccuracy. As to cubic-wise balance method, the distortions added to cells

are counterbalanced by their neighbor cells. Thus, zero-sum method and cubic-wise balance method performed much better for small size of queries than the random data distortion method.

The experiment results also showed that both zero-sum method and cubic-wise balance method could achieve better privacy than that of random data value distortion method in most cases. One of the common concerns among people is that what level of privacy would be reasonable for practical purpose. Generally speaking, a 100% privacy with 5% to 10% loss of accuracy was considered to be a satisfactory level.

As to the cubic-wise balance method and zero-sum method, zero-sum method can create the distorted data cube more efficiently than cubic-wise balance method, because the computing complexity of zero-sum method is $d * n_1 * n_2 * \cdots * n_d$ and the computing complexity of cubic-wise balance method is $2^d * n_1 * n_2 * \cdots * n_d$, where d is the number of dimensions of the data cube, n_i is the size of ith dimension. But cubic-wise balance method performs better than zero-sum method in terms of privacy and accuracy. Furthermore, besides the two parameters, cube sparsity and data distortion range, which would affect the performance, zero-sum method has an extra parameter: block size. The experiment results indicate that there was a trade-off between privacy and accuracy with different block size. The smaller that block size was, the higher accuracy the zero-sum method could obtain but the lower privacy it resulted in. How to select a suitable block size to obtain both better privacy and better accuracy is not clear so far. However, with cubic-wise balance method, users do not have to bother with deciding the size of block.

Since the response time in OLAP is critical, accessing distorted data instead of original data can not only protect the privacy of individual data but also introduce no restriction on data access. To further shorten the response time, we material-

ized a data cube in a multi-dimensional database. Harinaryan et al[46] stated that the materialized data cube retain a significant performance advantage, but it also raises the concern of scalability because the computation cost of zero-sum method or cubic-wise balance method increased dramatically with the increase of the number of dimensions. So the cost was relatively high for large data cubes.

However, this was only one time expense for a data cube, and the subsequent query response time was not affected. Furthermore, once the distorted data cube was created, the update cost was very low. Only the block which contains the updated cell needed to be recomputed. For OLAP applications, the query response time is more critical. In contrast, many other privacy preserving methods, such as query restriction, required extra processing time each time when answering a query. This additional processing time might significantly increase the response time.

In fact, the similar scalability problem also exists in pre-computing [47]. For example, the time of pre-computing may stretch to days in very large cases[48]. Parallel computing is one of the general techniques which has been used to efficiently handle the increase in data sizes [49][50][51]. However, the scalability of the cubic-wise balance method should be further considered in the future work.

We have also tried to extend the two methods to aggregated range-max queries and the results are not bad. But the results are not as good as range-sum queries. Thus further study is still needed to improve the accuracy.

Bibliography

[1] S.Chaudhuri, U.Dayal, An overview of data warehousing and olap technology, SIGMOD Record 26 (1) (1997) 65–74.

[2] J.Gray, A.Bosworth, A.Layman, H.Pirahesh, Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total, in: Proc. of 12th International Conference on Data Engineering, 1996, pp. 152–159.

[3] R.Agrawal, A.Gupta, S.Sarawagi, Modeling multidimensional databases, in: Proc. of the 13th International Conference on Data Engineering, Birmingham, U.K, 1997, pp. 232–243.

[4] C.T.Ho, R.Agrawal, N.Megiddo, R.Srikant, Range queries in olap data cubes, in: SIGMOD Tucson, 1997, pp. 73–88.

[5] S.Y.Lee, T.W.Ling, H.G.Li, Hierarchical compact cube for range-max queries, in: Proceedings of the 26th International Conference on VLDB, Cairo, Egypt, 2000, pp. 232–241.

[6] Hackers, Crackers, Spooks, Ensuring that your data warehouse is secure, DBMS Magazine 10 (4) (1997) 14.

[7] D.Barbara, M.Sullivan, Quasi-cubes: exploiting approximations in multidimensional databases, ACM SIGMOD Record 26 (3) (1997) 12–17.

[8] S.Chaudhuri, K. Shim, Including group-by in query optimization, in: In Proc. of the 20th Int'l Conference on Very Large Databases, 1994, pp. 354–366.

[9] A.Gupta, V.Harinarayan, D.Quass, Aggregate-query processing in data warehousing environments, in: In Proceedings of the Eighth international Conference on Very Large Databases(VLDB), 1995, pp. 358–369.

[10] P.Y.Weipeng, L. Per-A*ke, Eager aggregation and lazy aggregation, in: Proceedings of the 21th International Conference on Very Large Data Bases table of contents, 1995, pp. 345–357.

[11] S.Agarwal, R.Agrawal, P.M.Deshpande, A.Gupts, J.F.Naughton, R.Ramakrishman, S.Sarawagi, On the computation of multidimensional aggregates, in: In Proc. of the 22nd Int Conference on Very Large Database, 1996, pp. 506–521.

[12] G.Himanshu, H.Venky, R.Anand, D.U.Jeffrey, Index selection for olap, in: Proceedings of the Thirteenth International Conference on Data Engineering, 1997, pp. 208–219.

[13] A.Shoshani, Olap and statistical databases: Similarities and differences, in: Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson Arizona, USA, 1997, pp. 185–196.

[14] N. R.Adam, J. C.Wortman, Security-control methods for statistical databases: A comparative study, ACM computing surveys 21 (4) (1989) 515–556.

[15] I.P.Fellegi, On the questions of statistical confidentiality, J. Am. Stat. Assoc. 67 (337) (1972) 7–18.

[16] D.E.Denning, P.J.Denning, M.D.Schwartz, The tracker: A threat to statistical database security, ACM Transactions on Database Systems 4 (1) (1979) 76–96.

[17] D.Dobkin, A.K.Jones, R.J.Lipton, Secure database: Protection against user influence, ACM Transactions on Database Systems 4 (1) (1979) 97–106.

[18] D.E.Denning, Cryptography and Data Security, Addison-Wesley, 1982.

[19] R.Conway, D.Strip, Selective partial access to a database, in: Proceedings of ACM Annual Conference, 1976, pp. 85–89.

[20] V.Estivill-castro, L.Brankovic, Data swapping: Balancing privacy against precision in mining for logic rules, in: Proceedings of International Conference of Data Warehousing and Knowledge Discovery, Florence, Italy, 1999, pp. 389–398.

[21] J.F.Traub, Y.Yemini, H.Woznlakowski, The statistical security of a statistical database, ACM Transactions on Database System 9 (4) (1984) 672–679.

[22] L.L.Beck, A security mechanism for statistical databases, ACM Transactions on Database Systems 5 (3) (1980) 316–338.

[23] D.E.Denning, Secure statistical databases with random sample queries, ACM Transactions on Database Systems 5 (3) (1980) 291–315.

[24] R.Agrawal, R.Srikant, Privacy-preserving data mining, in: Proceedings of the nineteenth ACM SIGMOD conference on Management of Data, 2000, pp. 439–450.

[25] A.Evfimievski, J.Gehrke, R.Srikant, Limiting privacy breaches in privacy preserving data mining, in: Proc. of the Twenty-second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 2003, pp. 211–222.

[26] R. A.Evfimievski, R.Srikant, J.Gehrke, Privacy preserving mining of associations rules, in: Proceedings of the Eighth Conference on Knowledge Discovery and Data Mining, 2002, pp. 217–228.

[27] D.Agrawal, C.C.Aggarwal, On the design and quantification of privacy preserving data mining algorithms, in: Proc. of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 2001.

[28] I.Dinur, K.Nissim, Revealing information while preserving privacy, in: Proceedings of the Twenty-second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 2003, pp. 202–210.

[29] J.Vaidya, C.Clifton, Privacy preserving association rule mining in vertically partitioned data, in: Proceedings of the Eighth International Conference on Knowledge Discovery and Data mining, 2002, pp. 639–644.

[30] S.Rizvi, J.R.Haritsa, Maintaining data privacy in association rule mining, in: Proceedings of the Twenty-eighth Conference on Very Large Data Base, 2002, pp. 682–693.

[31] M.Kantarcioglu, C.Clifton, Privacy-preserving distributed mining of association rules on horizontally partitioned data, in: The ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Madison, Wisconsin, 2002, pp. 24–31.

[32] T.Priebe, G.Pernul, Towards olap security design - survey and research issues, in: Proceedings of the third ACM Int'l workshop on Data warehousing and OLAP, 2000.

[33] L.Wang, D.Wijesekera, S.Jajodia, Cardinality-based inference control in sum-only data cubes, in: Proc. of the 7th European Symposium on Research in Computer Security, 2002, pp. 55–71.

[34] H.Kargupta, S.Datta, Q.Wang, K.Sivakumar, On the privacy preserving properties of random data perturbation techniques, in: Proc. of 2003 IEEE International Conference on Data Mining. Melbourne, Florida, 2003, pp. 99–106.

[35] O. Council, Olap council's release ii of the analytical processing benchmark (apb-1) for olap server performance, in: http://www.olapcouncil.org/news/APB1r2b_PR.htm, 1998.

[36] R.Motwani, P.Raghavan, Randomized Algorithms, Cambridge Univ. Press., 1995.

[37] C.Faloutsos, H.Jagadish, N.Sidiropoulos, Recovering data from summary information, in: Proc. of the 23rd Int'l Conf. on Very Large Data Bases, 1997, pp. 36–45.

[38] D. Barbara, W. DuMouched, C. Faloutsos, P. Haas, H. J.M., Y. Ioannidis, The new jersey data reduction report, Data Engineering Bull 20 (1997) 3–45.

[39] J.Poland, A.Zell, Main vector adaptation: A cma variant with linear time and space complexity, in: In Proceedings of the Genetic and Evolutionary Computation Conference, 2001, pp. 1050–1055.

[40] B.Srinivasan, Parallel searching in distributed databases, Computer Networks 4 (1980) 157–166.

[41] Y.K.Lee, K.Y.Whang, Y.S.Moon, I.Y.Song, An aggregation algorithm using a multidimensinal file in multidimensional olap, Information Science 152 (1) (2003) 121–138.

[42] J.Gray, A.Bosworth, A.Layman, H.Pirahesh, Data cube: a relational aggregation operator generalizing group-by, cross-tabs and subtotals, in: In 12th IEEE Internatinal Conference on Data Engineering, 1995, pp. 152–159.

[43] M.Gyssens, L.V.S.Lakshmanan, I. Subramanian, Tabkes as a paradigm for quering and restructuring, in: ACM-PODS, 1996, pp. 93–103.

[44] C.Li, X.S.Wang, A data model for supportin on-line analytical processing, in: In proceedings of Conference on Information and Knowledge Management, 1996, pp. 81–88.

[45] E.Pourabbas, M.Rafanelli, Hierarchies and relative ooperators in the olap environment, ACM SIGMOD 29 (1) (2000) 32–37.

[46] V.Harinarayan, A.Rajaraman, J.D.Ullman, Implementing data cubes efficiently, in: Proc of the fifteenth ACM SIGMOD int'l conf. on Management of data, 1996, pp. 205–216.

[47] Y.Chen, F.Dehne, T.Eavis, A.Rau-Chaplin, Building large rolap data cubes in parallel, in: Proceedings of the International Database Engineering and Applications Symposium, 2004, pp. 367–377.

[48] Microsoft, EMC, Unisys, T3 project technical overview, White Paper .

[49] F.Dehne, T.Eavis, S.Hambrusch, A.Rau-chaplin, Parallelizing the data cube, Distributed and Parallel Databases 11 (2) (2002) 181–201.

[50] H.Lu, X.Huang, Z.Li, Computing data cubes using massively parallel processors, in: Proceedings of 7th Parallel Computing Workshop, 1997.

[51] S.Goil, A.Choudhary, High performance olap and data mining on parallel computers, Journal of Data Mining and Knowledge Discovery 1 (4) (1997) 391–417.